End-to-end Window-Constrained Scheduling for Real-Time Communication

Yuting Zhang and Richard West

Computer Science Department
Boston University
Boston, MA 02215
{danazh,richwest}@cs.bu.edu

Abstract. This paper extends our original work on window-constrained scheduling, to address the problem of meeting end-to-end service guarantees across a sequence of servers. We describe an algorithm, called Multi-hop Virtual Deadline Scheduling (MVDS), that attempts to minimize end-to-end window-constraint violations, while maximizing link utilization for a series of real-time streams. The challenge posed by the multi-hop problem is how to derive a local scheduling and dropping scheme from global service requirements, so that each server along a path can cooperate to guarantee end-to-end service. Similar to our VDS algorithm developed for a single server, MVDS orders packets at the heads of streams according to their *local* virtual deadlines. Using various packet dropping schemes at each server, based on current workloads and likelihoods of meeting end-to-end service constraints, we evaluate the performance of MVDS. Simulation results show that MVDS can provide better window-constrained service guarantees than other related algorithms, while still maintaining high link utilization.

1 Introduction

In the past few years, streaming multimedia applications have become popular among Internet users. Applications such as Windows Media Player and RealNetworks RealPlayer are commonly used to play back audio and video broadcasts on desktop PCs. End-system buffering is typically used to eliminate playback jitter as a result of network delays. However, this will not be the solution for next-generation distributed applications, involving the delivery of many thousands of live, and potentially high-bandwidth data streams. For example, applications such as live webcasts, interactive distance learning, tele-medicine and multi-way video conferencing may require the real-time capture of data, that must be transferred across a network in keeping with multiple end-user QoS requirements. Similarly, data aggregation applications (e.g., used in surveillance or traffic management) are now being developed on large-scale networks of sensors to capture and deliver QoS-constrained data to specific hosts, wireless devices and actuators [1].

Appropriate service disciplines are needed at various points along a network path, to meet the end-to-end requirements of real-time data streams. Moreover, the packets in such streams need to be scheduled by specific deadlines at each hop (e.g., a router, or end-host in an overlay network) to satisfy end-to-end service constraints. While many researchers have focused on end-to-end delay guarantees of *all* packets in a real-time stream [2, 3], there are many applications that only require timely and predictably service to a *fraction* of packets. Many multimedia applications can tolerate some packets being discarded or serviced late, as long as the consecutive number of such packets is limited. For example, a streaming video application might experience an acceptable reduction in signal-to-noise ratio rather than picture breakup, if bursts of late or lost packets are bounded.

To deal with the above classes of applications, we have developed several window-constrained scheduling algorithms [4–7], that provide weakly-hard [8, 9] service guarantees. Both our DWCS (Dynamic Window-Constrained Scheduling) and VDS (Virtual Deadline Scheduling) algorithms attempt to guarantee at least m out of a fixed *window* of k deadlines are met, for packets in real-time streams [1]. DWCS is capable of achieving 100% link utilization, while guaranteeing a feasible schedule for each stream, when all streams have the same request period (i.e., packet inter-arrival time). However, when streams have different request periods, DWCS may not generate a feasible schedule even when the link utilization is fairly low. For this reason, we developed VDS to meet window-constraints on streams with different packet inter-arrival times, or request periods.

Currently, we have only analyzed the performance of DWCS and VDS on single servers. However, most real-time traffic needs to cross multiple hops from the sender to the receiver and, therefore, end-to-end guarantees are required. The focus of this paper is on the multi-hop window-constrained scheduling problem: to ensure service constraints hold for all real-time streams across an ordered sequence of servers. Consequently, we extend our original window-constrained scheduling model to encompass end-to-end requirements. We use VDS as the basis for our multi-hop scheduling algorithm, as VDS has been shown to outperform DWCS on a single server for many scenarios.

Contributions: The paper describes an extension to VDS, called Multi-hop Virtual Deadline Scheduling (MVDS), that attempts to minimize end-to-end window-constraint violations of real-time streams while maximizing link utilization. Per-stream QoS requirements are specified across an entire path from source to destination, but scheduling decisions are made locally at each hop (or intermediate server). Without loss of generality, we assume there is no global control mechanism and no feedback signal from downstream hops to upstream hops in our model. The challenge is how to perform local scheduling decisions to meet global window-constraint service guarantees.

In MVDS, each server schedules a packet at the head of a stream according to its local *virtual* deadline. Each local virtual deadline is derived from a corresponding *real-time* deadline and window-constraint. By transforming end-to-end

[1] DWCS and VDS have both been applied to servicing periodic real-time threads, as well as packets in real-time streams.

deadlines and window-constraints into local (per server) values, real-time guarantees can be met across entire paths of servers. To show the effectiveness of our approach, we evaluate the MVDS algorithm using the NS simulator [10].

The rest of the paper is organized as follows: in the next section, we define the end-to-end window-constrained scheduling problem. The MVDS algorithm and several different schemes for dealing with late packets are then discussed in Section 3. In Section 4, we evaluate the performance of MVDS using alternative packet dropping and prioritization schemes. This is followed by a description of related work in Section 5. Finally, conclusions and future work are described in Section 6.

2 End-to-end Window-Constrained Scheduling Problem

Earliest deadline first scheduling (EDF) is capable of meeting all deadlines, when it is theoretically possible to do so on a single server. However, when a server is overloaded, it is impossible for *any* schedule to meet every deadline. If all deadlines cannot be met, window-constrained scheduling can still meet a minimum of m_i out of k_i deadlines for stream S_i, as long as no more than 100% of resources are required. In prior work, we have derived the conditions under which it is possible to make window-constrained service guarantees on a single server. In contrast, this paper focuses on the multi-hop window-constrained scheduling problem. Formally, we characterize a stream S_i by a six-tuple $(p_i, T_i, D_i, L_i, m_i, k_i)$, where p_i is the packet size, T_i is the period (or packet inter-arrival time at the first hop), D_i is the end-to-end delay bound, L_i is the number of hops (or servers) along an end-to-end path, and (m_i, k_i) is the window-constraint.

We assume every packet in S_i is the same size, p_i. For a stream with different size packets, we can use the maximum size as p_i. The maximum transmission delay (or service time) of a packet in stream S_i at each hop can be determined by the ratio of the (maximum) packet size, p_i, and the bandwidth of the link, B. We denote this transmission delay as C_i.

Each and every stream, S_i, is assumed to have periodic packet arrivals at the first hop, with the inter-arrival time, or request period, set to T_i. Packets can be made to arrive at fixed intervals at the first server using a periodic arrival process or a traffic shaper, such as leaky bucket. Hence, if $ts_{i,j}$ denotes the arrival time of jth packet of stream S_i at the first server, then the arrival time of the $j+1$th packet is $ts_{i,j+1} = ts_{i,j} + T_i$. However, the inter-arrival time of packets in S_i at other servers may be larger or smaller than T_i, due in part to scheduling effects.

The total end-to-end delay of a packet is the sum of the queuing delays, transmission delays and propagation delays along all L_i hops. For simplicity, we assume propagation delays are negligible. Similarly, we assume transmission delays are a constant, based on some fixed packet size and link bandwidth. The queuing delay of packets at each hop is dependent on the scheduling policy, and is considered the most important factor for determining end-to-end delay guarantees in this paper. Moreover, the end-to-end deadline of the jth packet of S_i is $ts_{i,j} + D_i$. In practice, $D_i \leq L_i T_i$ is typically true, but this is not a necessary requirement.

The window-constraint on S_i requires at least m_i out of every k_i consecutive packets to meet their end-to-end deadlines. That is, in every window of $(k_i - 1)T_i + D_i$ time, at least m_i packets arrive at the receiver (or final hop) within their delay bound D_i. When fewer than m_i out of a window of k_i packets meet their corresponding end-to-end deadlines, S_i is said to have violated its window-constraint. Here, the assumption is that each window of k_i packets is non-overlapping with respect to previous or successive windows in a given stream. This is different from the sliding window model used by [11]. In [4], we have shown that non-overlapping (or fixed) windows can be converted to sliding windows, and vice-versa. For example, for any fixed window-constraint, (m_i, k_i), the corresponding sliding window-constraint is $(m_i, 2k_i - m_i)$.

Given the above definitions, the end-to-end problem requires service guarantees on window-constrained streams, as they propagate sequentially from one server to the next. In this regard, we assume there is no global scheduling control and no feedback information from downstream servers. In summary, the problem is how to generate a schedule at each hop, that minimizes the number of window-constraint violations for all streams. Since any single hop cannot guarantee end-to-end service, all hops needs to coordinate their scheduling and dropping decisions. This is the motivation behind the extension to our earlier work on window-constrained scheduling.

3 Multi-hop Virtual Deadline Scheduling

To solve the end-to-end window-constrained scheduling problem, we propose an algorithm called Multi-hop Virtual Deadline Scheduling (MVDS). Since MVDS is based on our virtual deadline scheduler (VDS) for a single server, we begin by giving a brief overview of VDS.

3.1 Overview of VDS

In the window-constrained scheduling problem, each stream has two service requirements: a deadline and a window-constraint. VDS derives a "virtual deadline" from both requirements for each stream, and the stream with the earliest such deadline has the highest priority. In VDS, we assume every periodic real-time stream is serviced by a single server, and the request period of each stream is also its relative deadline, as defined in Rate Monotonic Scheduling [12]. One can think of a stream's request period as the interval between when its head packet is ready for service and when it must complete transmission. Likewise, a request period can be thought of as the inter-arrival time between successive packets in a given stream. We can now define a stream's virtual deadline with respect to real-time, t, as shown in Equation 1. The start time of the current request period for stream S_i at time t is $ts_i(t)$. In effect, this can be considered the arrival time of the latest packet is S_i. Similarly, (m'_i, k'_i) is the *current* window-constraint of the stream S_i.

$$Vd_i(t) = \frac{k'_i T_i}{m'_i} + ts_i(t),\ m'_i > 0 \tag{1}$$

Let us first outline the intuition behind a stream's virtual deadline. If, at time t, the current window-constraint of stream S_i is (m'_i, k'_i), then $m_i - m'_i$ out of $k_i - k'_i$ packets have been serviced in the current window. There are still m'_i packets that need to be transmitted, from the k'_i packets yet to arrive. It also follows that the remaining time in the current window is $k'_i T_i$. If in every interval $\frac{k'_i T_i}{m'_i}$ there is one packet transmitted from S_i, then m'_i packets can be serviced in the current remaining window-time, $k'_i T_i$. A side-effect of this is that S_i is assured proportional fairness guarantees with respect to other window-constrained streams. Additionally, delay jitter is minimized, by preventing at least m_i instances of S_i being serviced in a single burst at the end of a given real-time window.

Initially, the current window-constraint, (m'_i, k'_i), of each stream S_i is set to its original window-constraint (m_i, k_i). It gets updated according to the service that each stream initially requested and has currently received. Figure 1 shows the precise rules that dictate how the current window-constraint of each stream S_i is updated.

```
if a packet from stream $S_i$ gets service before its deadline,
    $m'_i = m'_i - 1$;

if a new packet arrives from stream $S_i$
    $k'_i = k'_i - 1$ ;
    if ($k'_i == 0$) {
        if ($m'_i > 0$) set tag as violation;
        $k'_i = k_i$; $m'_i = m_i$;
    }
```

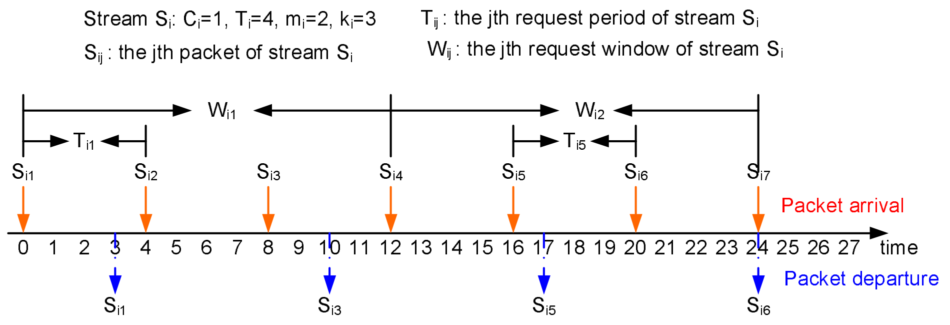
Fig. 1. Window-constraint Updates

As stated above, when a packet in S_i is serviced before its deadline, m'_i is decreased by 1, because fewer packets need to be serviced in current window. Thus, the priority of stream S_i is decreased and this consequently increases its virtual deadline. When a new packet arrives, k'_i is decreased by 1 because fewer packets will arrive in the current window. Under the assumption that packets arrive periodically for a given stream, there is a new packet arrival from S_i every request period T_i. Therefore, k'_i is updated every T_i, and the current remaining window time $k'_i T_i$ becomes shorter. The priority of stream S_i is implicitly increased as a result of its virtual deadline being reduced. When k'_i

reaches 0, it indicates the end of the current window. At this point, if m' is still larger than 0, a violation is set because there are less than m_i packets serviced before deadlines in the current window. Then the current window-constraint (m'_i, k'_i) is reset to its original value, (m_i, k_i).

Whenever a stream's current window-constraint is updated, its corresponding virtual deadline is recalculated. At each scheduling point, the packet with the earliest virtual deadline at the head of a stream, that is eligible for service, is chosen for transmission. However, not all streams are eligible at any time. To minimize the violation of all streams, a stream is precluded from a scheduling decision under the following two cases: 1) in each request period only one packet can be serviced, and a packet is dropped if it misses its deadline; 2) when m_i reached 0, then at least m_i packets in S_i have been serviced before their deadlines in the current window – in this case, S_i is given lower priority than any stream yet to meet its window-constraint. Only if all streams have achieved their minimum level of service can they again be considered in their current windows.

Figure 2 shows an example of how the current window-constraints and virtual deadlines are updated.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	t
2			1							0		2					1							0-2(reset)	m_i'
3				2				1				0-3(reset)				2				1				0-3(reset)	k_i'
0				4				8				12				16				20				24	ts_i
6				12				12				18				18				24				30	Vd_i

Fig. 2. Example of Virtual Deadline Calculation

In [7], we have shown by analysis and experiment that VDS can perform as well as our earlier DWCS algorithm, when the request periods of all streams are the same. Significantly, VDS can outperform DWCS when per-stream request periods are *different*. The reason is that deadlines and window-constraints are considered with equal importance in determining the scheduling priority. VDS combines these two parameters into a single metric, a *virtual deadline*, while DWCS uses them separately and orderly. Recent incarnations of DWCS order packets first in terms of their deadlines and, if there are ties, window-constraints are then considered. Hence, a packet with a long deadline in a stream with an urgent window-constraint requirement may be delayed longer than desired. Similarly, in the DBP algorithm [11], the scheduling priority is decided by the distance from a failing state, which is derived from the current window-constraint. As a consequence, DBP also suffers from similar problems to DWCS. In fact, we have shown in prior work that DWCS can perform at least as well as DBP [4], so we expect VDS to outperform DBP.

3.2 MVDS: Multi-hop Virtual Deadline Scheduling

For the single server scheduling problem, it is straight-forward to check whether a packet meets its deadline, and update the stream's current window-constraint. However, in the multi-hop case, any single server cannot determine whether end-to-end service guarantees will be satisfied. The challenge is how to derive a local scheduling and dropping scheme from the global service requirements, so that each server along a path can cooperate to guarantee end-to-end service. To solve the multi-hop window-constrained scheduling problem, we need to answer the following questions:

- How to transform end-to-end deadlines into local values for use at each hop?
- How to update current window-constraints and determine local scheduling states at each hop?
- How to strategically drop packets in order to minimize window-constraint violations, while maintaining high link utilization? In other words, how much queueing delay at each hop can be tolerated?

Local deadlines: In a multi-hop network, each stream may travel a different number of hops, and each hop may have different service demands. We should not use end-to-end deadlines alone to determine local scheduling priorities and state updates at each hop. Rather, local deadlines should be used by each server to generate a feasible end-to-end schedule.

A key property of multi-hop networks is that packet schedulers at downstream nodes (or servers) have an opportunity to compensate for excessive latencies due to congestion at upstream nodes. Similarly, downstream nodes may reduce the priority of packets receiving preferential service at prior, upstream nodes. To exploit such inter-node coordination, we adopt the same idea to determine local packet deadlines as in Coordinated Multi-hop Scheduling (CMS) [3]. The local deadline, $d^h_{i,j}$, of the jth packet in S_i at the hth hop is defined as follows:

$$d^h_{i,j} = \begin{cases} ts_{i,j} + \delta^1_{i,j}, & h = 1 \\ d^{h-1}_{i,j} + \delta^h_{i,j}, & 1 < h \leq L_i. \end{cases} \tag{2}$$

Here, $ts_{i,j}$ is the arrival time of the jth packet in S_i at the first hop. One can think of $ts_{i,j}$ as the timestamp of the packet. Also, $\delta^h_{i,j}$ is the relative delay bound at hop h. It is a non-negative function of i, h, p_i and D_i, and should satisfy $\sum_{h=1}^{L_i} \delta^h_{i,j} \leq D_i$. Hence, if a packet can meet its local deadline at each hop along the path, it must be able to meet its end-to-end deadline. In this equation, we can see the local deadline of each packet is not dependent on its arrival time at the current hop, but rather its local deadline at the previous hop. This means a server can compensate for variations in service at previous hops, and adjust the priority of each packet accordingly. One can think of a packet's previous local deadline as its logical arrival time at the current hop.

There are several different approaches to derive $\delta^h_{i,j}$. For simplicity, we use a constant value at each hop, such that $\delta^h_{i,j} = \frac{D_i}{L_i} \mid \forall h$ in our simulations described in Section 4. In general, other values for $\delta^h_{i,j}$ can be used. MVDS uses this local deadline at each server to update the corresponding window-constraint, and calculate virtual deadlines.

Local current window-constraints: The end-to-end (m, k) window-constraint requires at least m_i packets meet the (end-to-end) deadlines in every non-overlapping window of k_i consecutive packets, for each stream S_i. The current window-constraint should be updated according to the service it requires and has so far received in the current window. In the single-server case, VDS updates k'_i when a new packet arrives, and updates m'_i when a packet meets its deadline. However, in a multi-hop network, the traffic pattern may be affected in upstream servers. The original window of k_i consecutive packets at the first hop may not be maintained in the the following hops due to packet drops along a given path. Figure 3(a) shows the distortion to the original window of packets. To meet the end-to-end window-constraint for a stream, we should group the original k_i consecutive packets in the same window at each hop. Therefore, instead of updating window-constraints according to the locally arriving traffic, each hop should infer the original traffic at the first hop, and update window-constraints accordingly.

Without loss of generality, we assume the packets in each stream arrive at the *first* hop periodically, and are transmitted via a sequence of servers. We also assume that the sequence number and timestamp are recorded in the packet header, e.g. in an RTP header field. Hence, according to the sequence number or timestamp of an arriving packet, we can infer how many packets have been dropped in the previous hops, and how many packets have arrived in the current window. This means we can keep track of original windows of packets at each hop, and update local window-constraints accordingly. For example, if the sequence number of the head packet of S_i at hop h is j, then $k^{h'}_i$ is simply k_i - (j mod k_i).

It should be noted that a packet serviced by its local deadline at all hops along a path must be able to meet its end-to-end deadline. Therefore, we update $m^{h'}_i$ whenever a packet meets its local deadline. In effect, each server should attempt to meet m_i local deadlines in every window of the original k_i consecutive packets. However, even if a packet misses its local deadline at some hop, it may still be able to meet its end-to-end deadline at the receiver. We should not simply drop the packet if it only misses its local deadline. Rather we can derive a scheme for dropping packets at each server according to factors such as workload and the delay they have currently experienced. Before we describe the local drop scheme, Figure 3(b) shows how to maintain the correct window for distorted traffic at each hop.

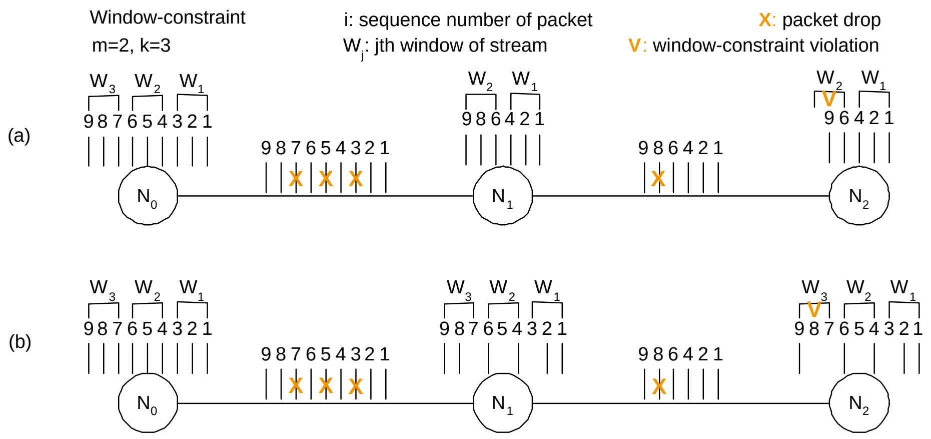


Fig. 3. Effects of Packet Dropping on Window-constraints at Successive Hops

Local drop scheme: When a packet misses its local deadline at some hop, it may make up for its tardiness at a downstream hop, so as to meet its end-to-end deadline. If we simply discard a packet, it may cause unnecessarily window-constraint violations and decrease link utilization. However, if we don't drop it, it may affect the service of other packets and/or potentially miss its end-to-end deadline. Especially in the overload case, when it is impossible to serve every packet in each stream, we must drop some packets. We assume that if a packet misses its end-to-end deadline, it is unnecessary to send it to its final destination. Therefore, whenever a packet misses its end-to-end deadline at any hop, it is always discarded. The bandwidth consumed by this packet transmitted from previous hops is nonetheless wasted. It follows that a useful service at hop, h, is one which contributes to a packet meeting its end-to-end service requirement. In order to maximize the link utilization, we want to minimize wasteful services and, consequently, the number of packets that are dropped. At the same time as maximizing resource usage, we still need to guarantee window-constraints where possible.

For those packets missing their local deadlines but not their end-to-end deadlines, we propose a probabilistic drop scheme. Each packet is dropped according to a certain probability, derived from a function of a server's minimum utilization and a packet's current delay. The minimum utilization at hop h is the minimum service that hop h must provide to all streams passing through it, in order to satisfy their window-constraints. It is defined as:

$$U^h_{min} = \sum_{\forall i \in S(h)} \frac{m_i C_i}{k_i T_i}, \mid S(h) = \{i \mid S_i \; passes \; through \; hop \; h\} \tag{3}$$

$S(h)$ is the set of all streams traveling through hop h. In order to meet the window-constraints, at most $1 - U_{min}$ wasted utilization can be tolerated for all streams. In general, for lower values of U_{min}, a packet has less probability of being dropped, even if it misses its local deadline. When U_{min} approaches 1.0, fewer packets can be serviced late because they impact service guarantees on other packets meeting their deadlines. Consequently, the drop probability, p, is a function of U_{min}, such that $p \propto \frac{1}{1-U_{min}}$ when $U_{min} < 1.0$. For values of U_{min} equal to 1.0, we cannot tolerate servicing any late packets in order to meet end-to-end window-constraints. Observe that for values of U_{min} larger than 1.0, it is impossible to generate a feasible window-constrained schedule.

To be fairer, we also consider the latency experienced by each packet when deciding its drop probability. The latency factor of the jth packet of stream i at hop h, at time t, is defined as:

$$\begin{aligned} lat^h_{i,j}(t) &= \frac{t - d^h_{i,j}}{\boldsymbol{D_{i,j}} - d^h_{i,j}} \\ \boldsymbol{D_{i,j}} &= ts_{i,j} + D_i \end{aligned} \tag{4}$$

$d^h_{i,j}$ and $\boldsymbol{D_{i,j}}$ are the local deadline at hop h, and the end-to-end deadline, respectively. The more latency a packet has experienced, the less chance it has of meeting its end-to-end deadline, and therefore it is more likely to be dropped.

To take in account both server utilization, U_{min} and packet delay, $lat^h_{ij}(t)$, the drop probability at time t, of packet j from stream i at hop h is defined in Equation 5.

$$p^h_{ij}(t) = \begin{cases} min(1, \alpha(\frac{1}{1-U_{min}} - (1 - lat^h_{ij}(t)))), & U_{min} < 1 \\ 1, & U_{min} = 1 \\ lat^h_{ij}(t) & U_{min} > 1. \end{cases} \tag{5}$$

α is a reference parameter for the function, such that $0 < \alpha \leq 1.0$. We use a value of 0.1 for α in our simulations described later. In the previous equation, we can see that in the under-load case, the drop probability mainly depends on the utilization, but in the over-load case it depends more on packet latency.

MVDS (Multi-hop Virtual Deadline): According to the above discussion, MVDS derives virtual deadlines for head packets in each stream, and uses them to decide scheduling order at each hop. Based on the VDS algorithm, the local virtual deadline of the jth packet at the head of each stream is calculated using Equation 6. The packet with the earliest local virtual deadline has the highest priority.

$$Vd^h_{i,j}(t) = \frac{k^{h'}_i \delta^h_{i,j}}{m^{h'}_i} + ts^h_{i,j} \quad (m^{h'}_i > 0) \tag{6}$$

In the above equation, $ts^h_{i,j}$ is latest time of arrival of the jth packet at hop h given that it meets its local deadline, $d^{h-1}_{i,j}$, at the previous hop. If several streams' local virtual deadlines are equal, then we use local deadlines and remaining path lengths to break the ties. Some properties of MVDS can be summarized as following, the proofs are omitted due to space limitations..

- A necessary (but not sufficient) condition for MVDS to provide window-constraint service guarantees to S_i is: $U^h_{min} = \sum_{\forall i \in S(h)} \frac{m_i C_i}{k_i T_i} \leq 1$ $(\forall h \in P(S_i))$. $P(S_i)$ is the set of hops in the path of stream S_i.
- Assuming $D_i \leq L_i T_i$, the maximum buffer size for each stream S_i at any hop in MVDS is $[1, \lceil \frac{D_i}{T_i} \rceil]$ packets. If packets are dropped only when they miss their local deadlines, the maximum buffer size for each stream is 1 packet. Similarly, if packets are dropped only when they miss their end-to-end deadlines, the maximum buffer size for each stream is $\lceil \frac{D_i}{T_i} \rceil$ packets.
- If S_i meets its end-to-end window-constraint, the maximum jitter (or delay variation) experienced by S_i at any hop h, is $2((k_i - m_i)T_i + D_i - C_i)$.
- The complexity of the MVDS algorithm is $O(n)$ per hop in the worst case, where n is the number of streams requiring service.

4 Experimental Evaluation

This section evaluates the performance of MVDS using a series of ns-2 simulations [10]. We consider a simple network topology as shown in Figure 4. All link rates are 10Mbps and all propagation delays are ignored. We assume all packet lengths are 1000 bytes. The main streams enter the network from the first node N_0 and exit the network from the last node, N_6. All main streams share the longest path comprising 6 hops. Additionally, window-constrained cross traffic occurs at each link, and shares only one hop with the main stream before exiting the network. We consider 12 scheduling classes for all streams. The first 6 classes of streams are main streams. Their original window-constraints expressed as a fraction (m/k) are 3/5, 5/7, 7/9, 9/11, 11/13, and 13/15, respectively. Corresponding arrival rates are 333kbps, 200kbps, 143kbps, 110kbps, 90kbps and 77kbps. All 6 remaining classes of streams represent cross traffic at respective hops along the main network path. Each cross traffic class has the same window-constraints and arrival rates as the corresponding main traffic class. For example, the first cross traffic class for stream s_7 has original window-constraints and arrival rates of 3/5 and 333kbps, respectively. We assume that each class has the same number of streams. We vary the total flow number from 72 to 156 to generate different workloads for the network. In each load case, we run the simulation up to 100 seconds.

To measure the effect of end-to-end deadlines on our algorithm, we consider two scenarios in the simulation:

- Scenario 1: We set the end-to-end delay bound, D_i, of each stream, S_i, to the value $L_i T_i$. Hence, with the above simulation settings, the end-to-end delay bounds of the 6 main stream classes are 144ms, 240ms, 336ms, 432ms, 528ms, 624ms. Likewise, the 6 classes of cross traffic streams have delay

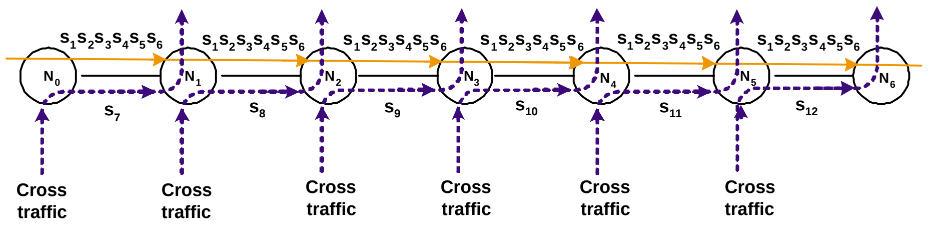


Fig. 4. Network Topology used for Simulations

bounds set to 24ms, 40ms, 56ms, 72ms, 88ms and 104ms respectively. We will refer to this scenario as the "long deadline" case.

- Scenario 2: We set the end-to-end delay bound, D_i, of each main stream, S_i, to the value $\frac{L_i}{2}T_i$, and the end-to-end delay bound, D_j, of each cross traffic stream, S_j, to the values in Scenario 1. Hence, with the above simulation settings, the end-to-end delay bounds of the 6 main stream classes are 72ms, 120ms, 168ms, 216ms, 264ms, 312ms. Likewise, the 6 classes of cross traffic streams have delay bounds as before. We will refer to this scenario as the "short deadline" case.

Performance Metrics: The traffic load in the network is represented by the maximum U_{min} across all hops (i.e., max $U_{min} = max\{U^1_{min} \cdots U^6_{min}\}$, where U^1_{min} is the minimum utilization requirement at N_0 etc). This represents the x-axis in all figures below. For all experiments, we measure the average miss rate and window-constraint violation rate per stream for different network loads. The miss rate is calculated as the ratio of the number of packets missing their end-to-end deadlines to the total number of packets sent from each stream's source. The window-constraint violation rate is calculated as the ratio of the number of windows with end-to-end violations to the total number of non-overlapping windows. In the following results, we show the performance across *all* streams, and then just the *main* streams when cross traffic is discounted. Additionally, we compare the performance of MVDS using different packet drop schemes, and also the performance of MVDS against other scheduling algorithms for both scenarios described above.

Performance of different drop schemes: Figures 5 and 6 show the performance of MVDS with different drop schemes in both scenarios, respectively. With the "no-drop" scheme, all packets in the network are serviced without dropping. With the "drop-end" scheme, a packet is dropped at any hop if it misses its end-to-end deadline. Similarly, with the "drop-local" scheme, a packet is dropped at any hop if it misses its local deadline. Finally, the "drop-prob" scheme causes a packet to be dropped according to its drop probability as described in the previous section. With the exception of the "no-drop" scheme all other schemes drop a packet if it has already missed its end-to-end deadline. Hence, the deadline miss rate is the same as drop rate.

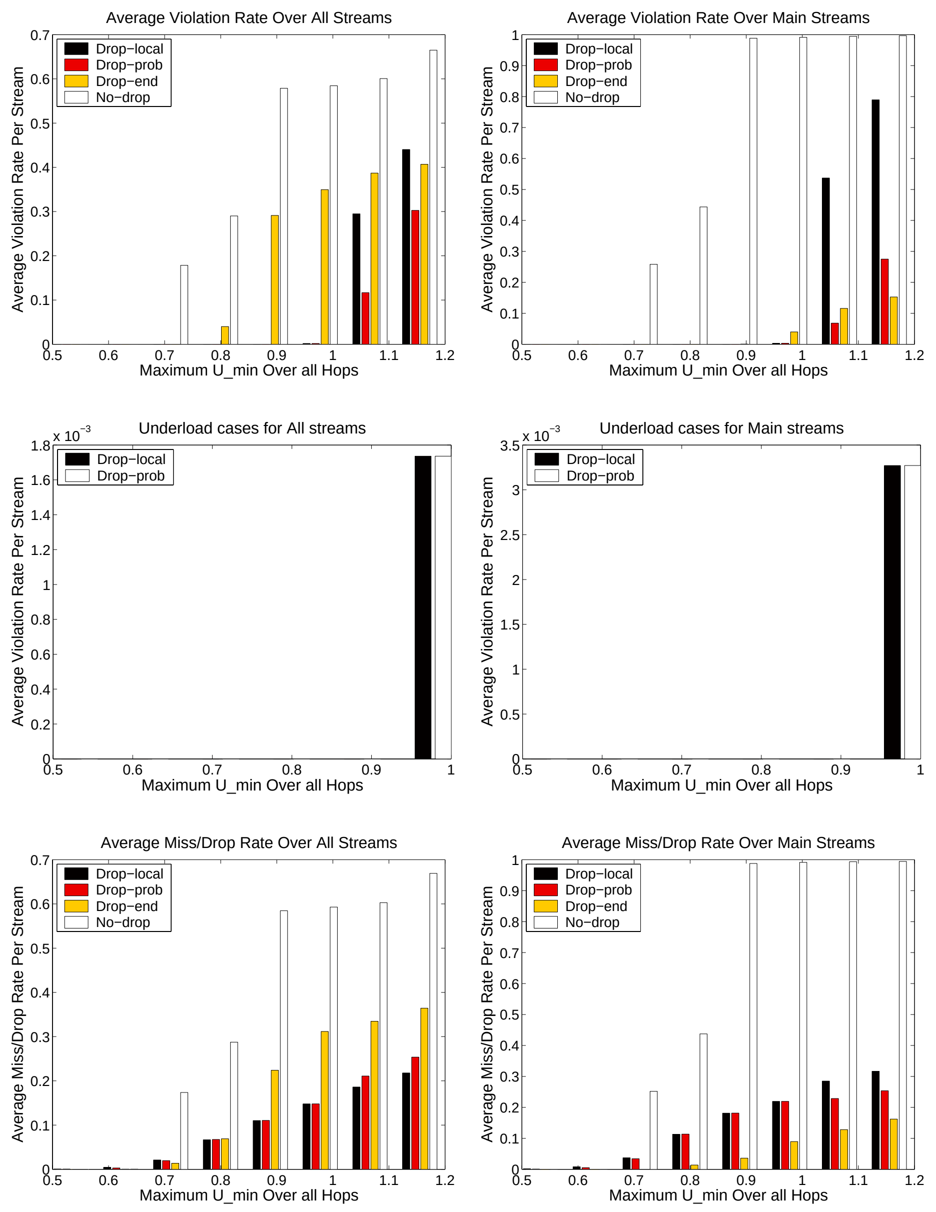


Fig. 5. Performance of MVDS with different drop schemes in Scenario 1

From the results, we can see the "no-drop" scheme performs worst, since the service of the previous packet will delay the service of all ensuing packets, and cause more deadline misses and window-constraint violations. The "drop-local" scheme yields a low violation rate in under-load situations, but may drop

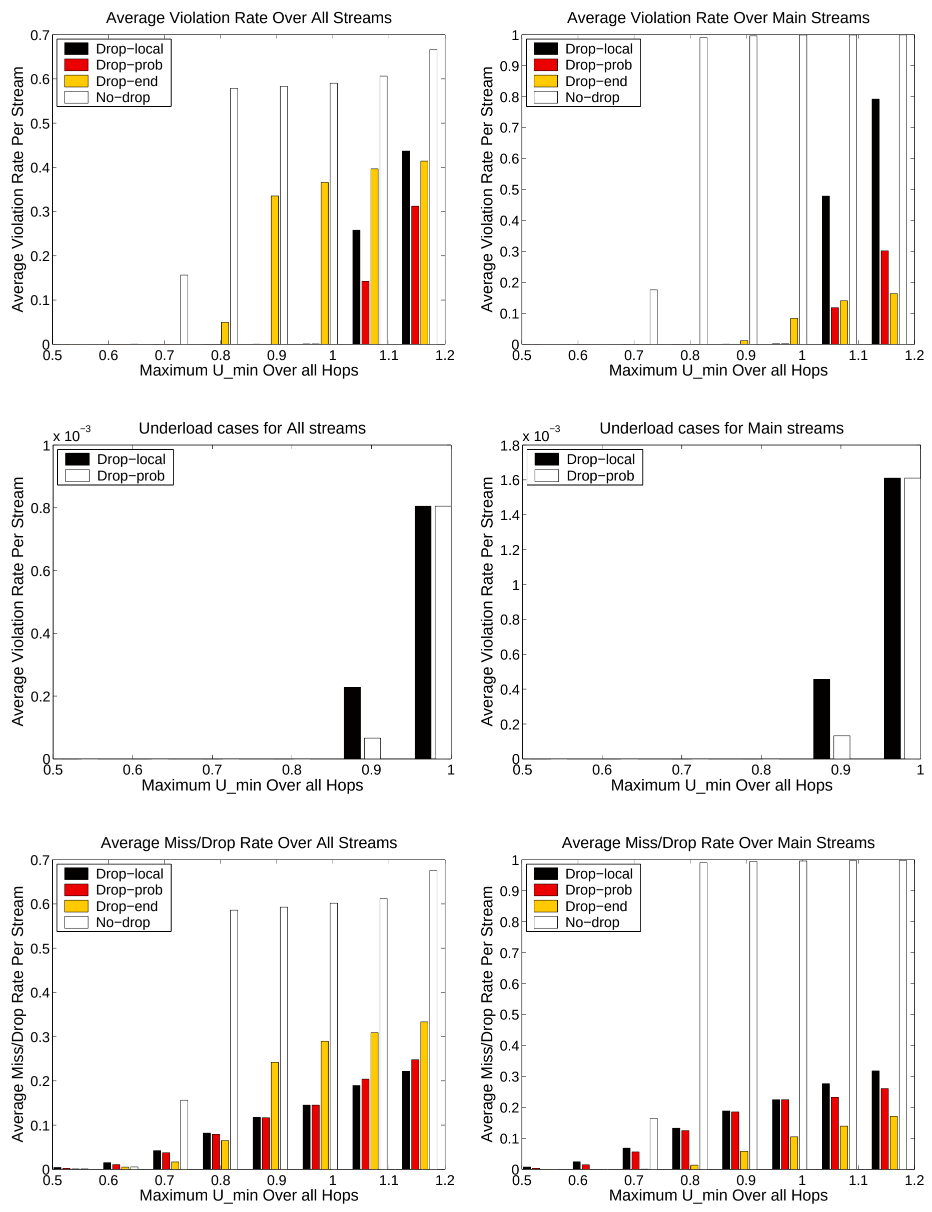


Fig. 6. Performance of MVDS with different drop schemes in Scenario 2

more unnecessary packets when loads are very small. Moreover, the "drop-local" scheme favors cross traffic streams with shorter path lengths than those of the main streams, because it drops packets very early. It can also cause more violations in the overload case. The "drop-end" approach produces low drop rates when load is small but, overall, causes more violations than the “drop local” approach. It favors main streams with the longer path lengths than those of cross traffic. Overall, we can see the “drop-prob” approach achieves the smallest violation rate, while keeping the miss rate low, in both under-load and over-load cases. “Drop-local” and “drop-prob” both begin to have violations when the load is about 1.0 in scenario 1, and when the load is about 0.9 in scenario 2. However, the violation rate of “drop-prob” is as low as 0.6×10^{-4} when the load is approximately 0.9, while the violation rate of the “drop-local” scheme is 2.3×10^{-4}. It should be noted that with the “drop-prob” scheme, window-constraint violations and packet drops are spread more evenly between cross traffic and main streams. The performance in Scenario 2 is slightly worse than in Scenario 1, although all drop schemes show the same relative performance.

Performance of different priority schemes: Figures 7 and 8 compare the performance of MVDS against other scheduling policies, including MDWCS and Coordinated EDF [3], respectively. Observe that MDWCS (a multi-hop version of our DWCS scheduling algorithm [4]) behaves in a manner similar to that of DBP-M [13]. Since we are mainly interested in the effects of packet ordering imposed by these algorithms, in terms of end-to-end service guarantees, we assume they all use the same packet dropping scheme. In all three algorithms, we use the “drop-prob” scheme. Although this is slightly different than the dropping schemes used in the original Coordinated EDF algorithm, and others such as DBP-M, it actually improves their performance.

In the case of MDWCS, current (local) window-constraints are first used to decide packet ordering. Any pair of packets having the same local window-constraints are then ordered based on their local deadlines at the current hop. This scheme is similar to DBP-M [13], a multi-hop scheduling approach that is based on the DBP algorithm [11]. In DBP-M, the distance from a (window-constraint) failing state is calculated according to state information concerning the number of packets in a stream’s recent history that have met their *local* deadlines. In prior work, we have shown that DWCS performs at least as good as DBP for a single server [4]. Consequently, we expect MDWCS to perform at least as well as DBP-M for a multi-hop network of servers.

In contrast to other algorithms, Coordinated EDF makes scheduling decisions using only the local deadlines of packets at the heads of streams. Compared to both CEDF and MDWCS, MVDS can achieve lower end-to-end window-constraint violation rates. In scenario 1, CEDF, MDWCS and MVDS begin to violate end-to-end service requirements when the network loads are about 0.7, 0.8 and 1.0, respectively. In Scenario 2, CEDF, MDWCS and MVDS begin to violate service requirements when the network load is about 0.7, 0.6 and 0.9, respectively. In both scenarios, the violation rate of MVDS is much less than MDWCS and CEDF. The reason is that MVDS combines the per-stream window-constraints and deadlines together, to decide the schedule priority, while other algorithms either only use one such service constraint (e.g., CEDF), or consider them separately (e.g., MDWCS).

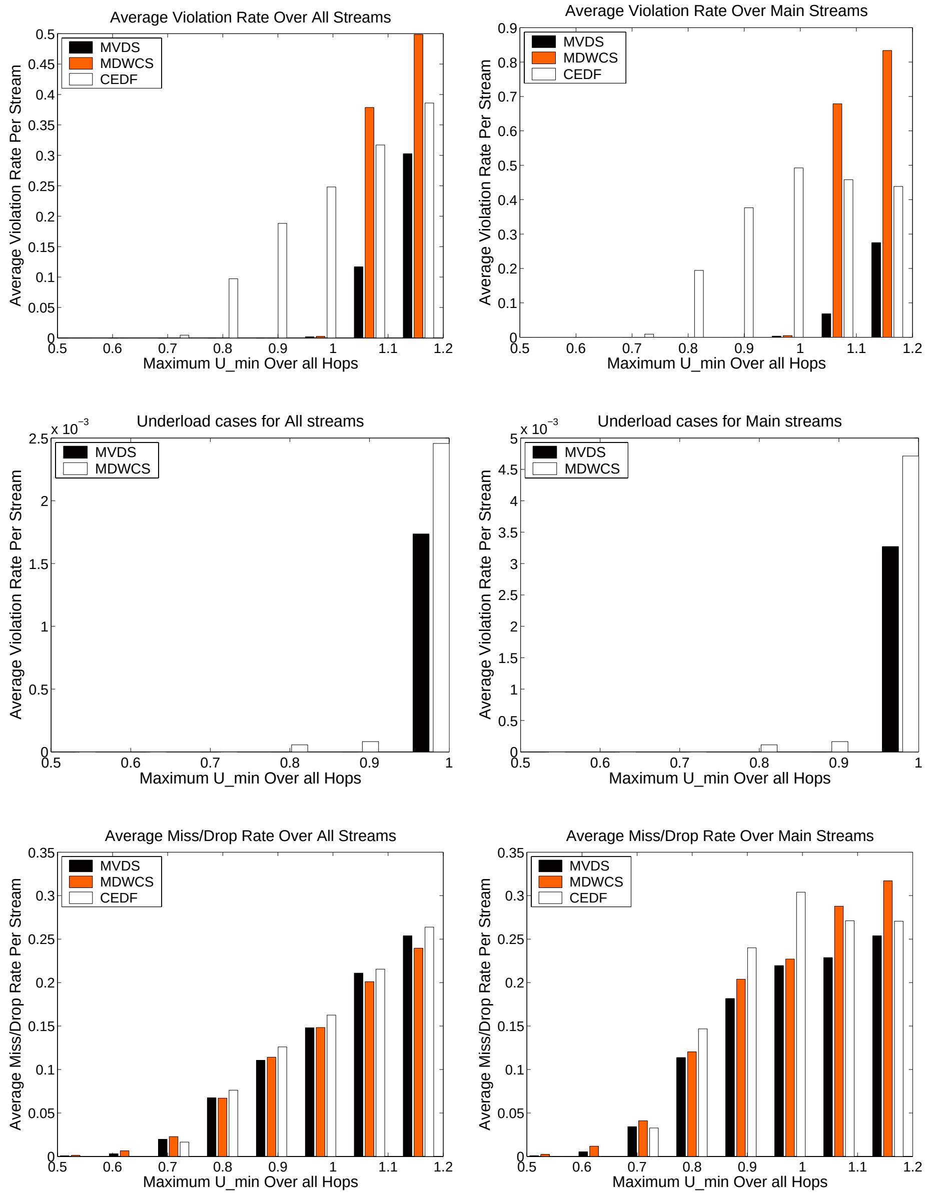


Fig. 7. Comparisons of MVDS, MDWCS, and CEDF in Scenario 1

5 Related Work

The problem of scheduling real-time messages in packet-switched networks has been studied extensively in recent years. Numerous algorithms have been proposed. Some are priority-based such as Delay-Earliest Due Date (D-EDD), Jitter-

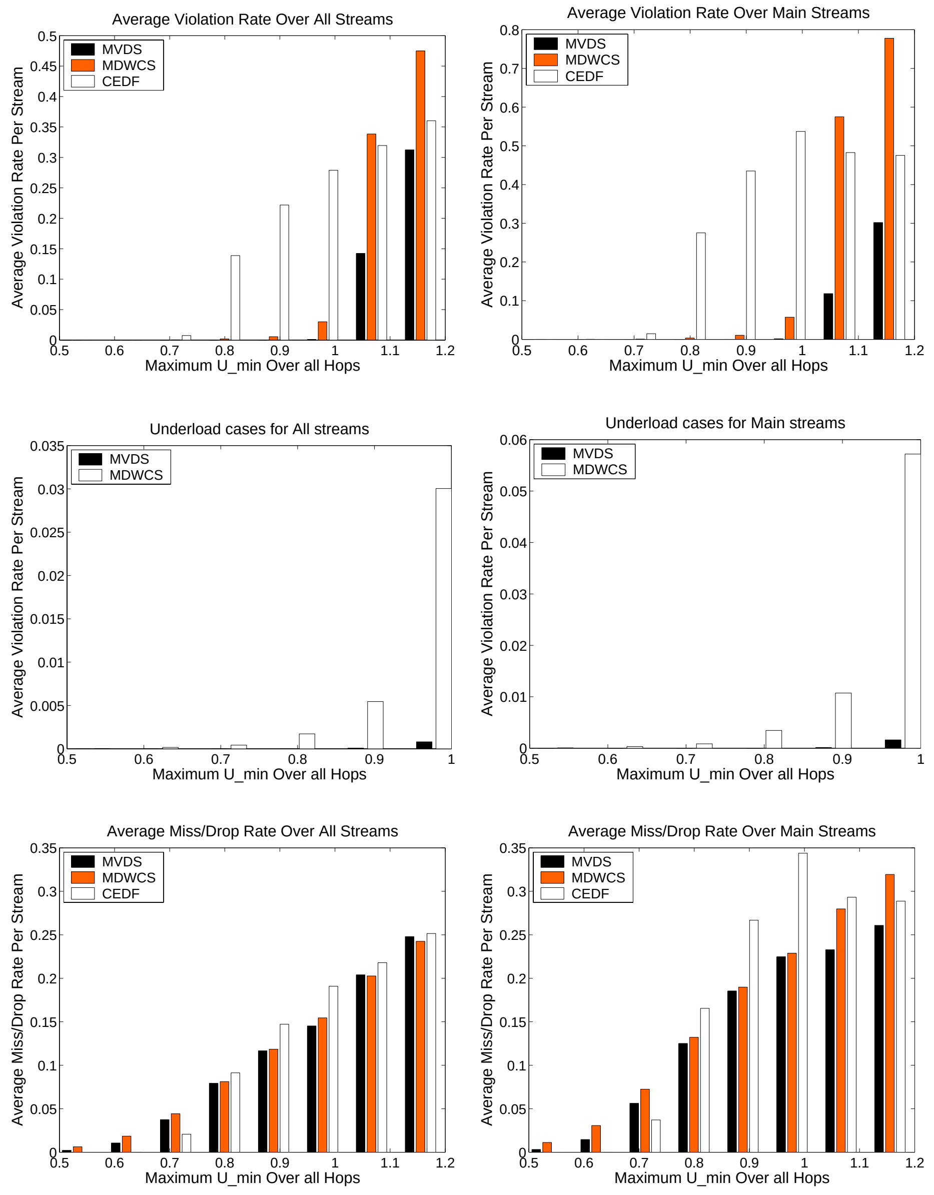


Fig. 8. Comparisons of MVDS, MDWCS, and CEDF in Scenario 2

Earliest Due Date (J-EDD), Virtual Clock (VC), and Fair Queueing (FQ) [14]. Some are channel-based such as Hierarchical Round-Robin (HRR) [14], and Budgeted-Weighted-Round-Robin (BWRR) [2]. However, none of these algorithms make use of cooperation between individual servers to ensure end-to-end guarantees. More precisely, downstream hops can compensate for excessive latency or unfairness incurred at upstream hops. Alternatively, downstream hops can reduce the priority of a packet which arrives ahead of schedule due to a lack of congestion upstream.

Some scheduling policies do consider the effects of service at prior hops in a communication path. These include Global Earliest-Deadline-First (G-EDF) [15], Coordinated Earliest-Deadline-First (CEDF) [16], and Modified First-In-First-Out [17]. In [3], they classified these schedulers and proposed a framework for design and analysis of multi-hop scheduling, that exploits inter-hop coordination. While such approaches try to provide hard real-time service guarantees, such that every packet meets its end-to-end delay bound, MVDS attempts to provide weakly-hard service.

Weakly-hard [8, 9] and (m, k)-firm scheduling [11] are both similar to our window-constrained scheduling schemes. Hamdaoui and Ramanathan [11] were the first to introduce the notion of (m, k)-firm deadlines, in which statistical service guarantees are applied to jobs. Their algorithm uses a "distance-based" priority (DBP) scheme to increase the priority of a job in danger of missing more than m deadlines over a sliding window of k requests for service. While our window-constrained scheduling algorithms all operate on fixed windows, they are nonetheless able to provide service guarantees over sliding windows as well. Moreover, our algorithms such as VDS [7] and DWCS [4] are provably superior to DBP in meeting (m, k) service requirements for a number of specific and non-trivial situations.

Similar to (m, k)-firm scheduling is the work by Koren and Shasha on 'skip over' scheduling [18]. Skip over scheduling allows certain job instances to be skipped, but may unnecessarily miss servicing a job instance when there are resources available. There are also examples of (m, k)-hard schedulers [19], but most such approaches require off-line feasibility tests, to ensure predictable service.

Finally, DBP-M is a multi-hop scheduling algorithm based on the Distance-based Priority (DBP) algorithm [13]. It uses a very simple heuristic method to decide per-server (local) deadlines, drop schemes and scheduling states. We have compared DBP-M and MVDS through analysis and simulation and believe that MVDS provides better service to window-constrained traffic with end-to-end requirements.

6 Conclusions and Future Work

In this paper, we address the end-to-end window-constrained scheduling problem, in which at least m out of (every window of) k consecutive packets should meet their end-to-end deadlines. Leveraging our prior work on virtual deadline scheduling (VDS) for a single server, we describe the multi-hop VDS algorithm (MVDS). By exploiting the cooperation between individual servers, we show how to transform global service constraints of real-time streams into localized values for use at each hop.

MVDS schedules head packets from different streams according to their deadlines and window-constraints *local* to the current server. Simulation results suggest that MVDS can outperform other methods which separately consider window-constraints and deadlines when prioritizing packets. Likewise, packet dropping schemes play an important role in determining the ability of a window-constrained scheduler to meet end-to-end guarantees. The probabilistic dropping scheme described in this paper outperforms alternative approaches, in its attempt to minimize service violation rates while maximizing link utilization.

MVDS is an heuristic algorithm providing (m, k)-firm end-to-end guarantees. Future work involves characterizing the probability that MVDS guarantees service to real-time streams. While MVDS requires $O(n)$ scheduling costs with respect to n streams, we are also interested in analyzing the performance of lower complexity algorithms where scale is a significant issue. Notwithstanding, MVDS will be used as part of our ongoing work to build an Internet-wide distributed system for processing and delivering real-time data streams to potentially many thousands of end-users. In this system, we envision the use of off-the-shelf PCs to form an overlay network for transporting information. MVDS will play an important role in the timely delivery of data between end-systems, even though we do not have total control of Internet traffic.

References

1. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “TAG: A tiny aggregation service for ad-hoc sensor networks,” in *Proceedings of Operating Systems Design and Implementation*, USENIX, December 2002.
2. I. Philip and J. Liu, “End-to-end scheduling in real-time packet-switched networks,” in *International Conference on Network Protocols (ICNP ’96)*, (Columbus, Ohio), October 1996.
3. C. Li and E. W. Knightly, “Coordinated multihop scheduling: a framework for end-to-end services,” *IEEE/ACM Transactions on Networking (TON)*, Dec. 2002.
4. R. West, Y. Zhang, K. Schwan, and C. Poellabauer, “Dynamic window-constrained scheduling of real-time streams in media servers,” *IEEE Transactions on Computers*, vol. 53, pp. 744–759, June 2004.
5. R. West and C. Poellabauer, “Analysis of a window-constrained scheduler for real-time and best-effort packet streams,” in *Proceedings of the 21st IEEE Real-Time Systems Symposium*, December 2000.
6. R. West, K. Schwan, and C. Poellabauer, “Scalable scheduling support for loss and delay constrained media streams,” in *Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium*, IEEE, June 1999.
7. Y. Zhang, R. West, and X. Qi, “A virtual deadline scheduler for window-constrained service guarantees,” Tech. Rep. 2004-013, Boston University, March 2004.
8. G. Bernat, A. Burns, and A. Llamosi, “Weakly-hard real-time systems,” *IEEE Transactions on Computers*, vol. 50, pp. 308–321, April 2001.
9. G. Bernat and R. Cayssials, “Guaranteed on-line weakly-hard real-time systems,” in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, December 2001.
10. “NS simulator: http://www.isi.edu/nsnam/ns/.”

11. M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm deadlines," *IEEE Transactions on Computers*, April 1995.
12. C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, January 1973.
13. W. Lindsay and P. Ramanathan, "DBP-M: A technique for meeting end-to-end (m,k)-firm guarantee requirements in point-to-point networks," in *Proceedings of the 22th IEEE Conference on Local Computer Networks(LCN'97)*, (Minneapolis, MN), Nov. 1997.
14. H. Zhang and S. Keshav, "Comparison of rate-based service disciplines," in *Proceedings of ACM SIGCOMM*, pp. 113–121, ACM, August 1991.
15. T. Chen, J. Walrand, and D. Messerschmitt, "Dynamic priority protocols for packet voice," *IEEE Journal on Selected Areas in Communications*, June. 1989.
16. M. Andrews and L. Zhang, "Minimizing end-to-end delay in high-speed networks with a simple coordinated schedule," in *IEEE INFOCOM'99*, (New York, NY), March. 1999.
17. D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network," in *ACM SIGCOMM'92*, (Baltimore, Maryland), Aug. 1992.
18. G. Koren and D. Shasha, "Skip-over: Algorithms and complexity for overloaded systems that allow skips," in *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pp. 110–117, IEEE, December 1995.
19. G. Bernat and A.Burns, "Combining (n/m)-hard deadlines and dual priority scheduling," in *Proceedings of the 18th IEEE Real-Time Systems Symposium*, (San Francisco), pp. 46–57, December 1997.