Technical Report

Schedulability Analysis of Generalized Multiframe Traffic on Multihop-Networks Comprising Software-Implemented Ethernet-Switches

Björn Andersson

HURRAY-TR-080201

Version: 1

Date: 04-02-2008

Schedulability Analysis of Generalized Multiframe Traffic on Multihop-Networks Comprising Software-Implemented Ethernet-Switches

Björn Andersson

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: bandersson@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

Consider a multihop network comprising Ethernet switches. The traffic is described with flows and each flow is characterized by its source node, its destination node, its route and parameters in the generalized multiframe model. Output queues on Ethernet switches are scheduled by static-priority scheduling and tasks executing on the processor in an Ethernet switch are scheduled by stride scheduling. We present schedulability analysis for this setting.

Schedulability Analysis of Generalized Multiframe Traffic on Multihop-Networks Comprising Software-Implemented Ethernet-Switches

Björn Andersson
IPP Hurray Research Group
Polytechnic Institute of Porto, Portugal
bandersson@dei.isep.ipp.pt

Abstract

Consider a multihop network comprising Ethernet switches. The traffic is described with flows and each flow is characterized by its source node, its destination node, its route and parameters in the generalized multiframe model. Output queues on Ethernet switches are scheduled by static-priority scheduling and tasks executing on the processor in an Ethernet switch are scheduled by stride scheduling. We present schedulability analysis for this setting.

Introduction

The Internet is undergoing two important changes. Organizations are replacing IP-routers with Ethernet switches; the trend is toward entire organizations having networks with Ethernet switches and only an IP-router for communication outside the network. In addition, interactive multimedia traffic such as Voice-over-IP and video-conferencing are increasingly popular and this brings the need to satisfy real-time requirements. A good illustration of this need is the fact that a large multinational company, delivering Voice-over-IP over Ethernet used in medical care in southern Sweden, has now been reported to the government for jeopardizing patient safety because network delays were too large [1].

Clearly a low delay is desired. The limited speed of light causes significant delays for traffic over large geographical distances; this cannot be reduced with better networking equipment. The delay due to queuing of a packet because other less time-critical packets are ahead in a queue can however be controlled by networking equipment and this is the subject of this paper.

Hops in the core of the Internet tend to have small queuing delay because of overprovisioning. The traffic in the core is an aggregation of a large number of independent flows and hence (due to the law of large numbers) the delay in the core has low variance as well; consequently an upper bound on the delay of hops in the core network can be estimated from measurements. Practitioners have therefor suggested that QoS techniques are most useful in the edge of the Internet [2, 3].

The edge of the Internet is heavily reliant on Ethernet technology and prioritized switches are becoming common there. Typically, a higher priority is given to Ethernet frames from one incoming interface or Ethernet frames carrying voice but unfortunately those networks do not use scheduling theory in order to find an upper bound on the delay. It is our belief however that schedulability analysis now has the potential to play an important role in the edge of the Internet because (i) Ethernet switches are based on point-to-point communication and hence there are no problems with random backoffs in the medium access as was the case in traditional shared-coaxial-cable/hub-based Ethernet in the past, (ii) queuing delays in outgoing queues in Ethernet switches can be controlled with static-priority scheduling according to the IEEE 802.1p standard, where a specific frame-format of the Ethernet frame specifies the priority, (iii) many commercially available Ethernet switches support 2-8 priority levels and can operate according to the IEEE 802.1p standard and (iv) many networking applications today need to meet deadlines.

Given the capability of current infrastructure and application needs it is worthwhile to develop architectures for achieving real-time guarantees on the Internet. Such architectures are well-explored (RSVP [4] is one of them) but they did not achieve widespread adoption. It is our belief however that offering real-time guarantees in the edge of the Internet and also in internal corporate networks and metropolitan networks is easier to adopt because it is typically owned by a single organization and hence it brings simplifications such as (i) the resource reservation (as a result of a flow being accepted by an admission test) can be performed without billing and (ii) complete knowledge of topology is possible.

Proving an upper bound on the end-to-end delay requires that pipelines of resources are analyzed. For this purpose, the real-time computing community has proposed a framework, called *holistic schedulability analysis* [5] which has been used successfully in

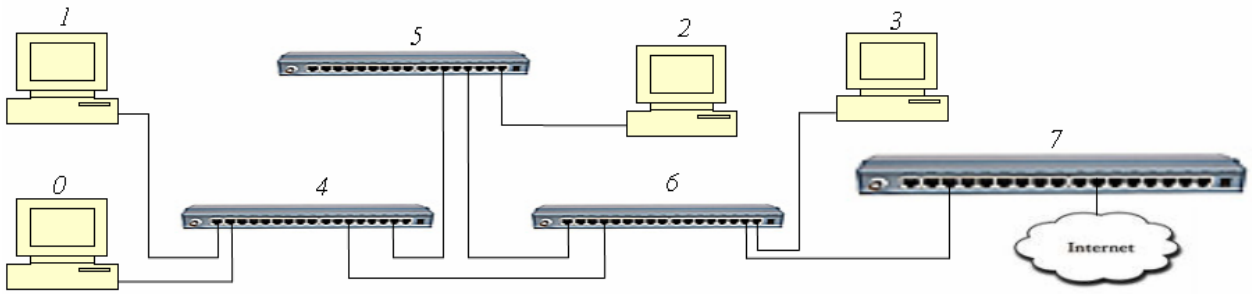


Figure 1. An example network with Ethernet switches. Node 0,1,2 and 3 are IP-endhosts, for example normal PCs running video conferencing applications. Node 4,5 and 6 are Ethernet switches. Node 7 is an IP-router which connects the Ethernet network to the global Internet.

automotive systems but as far as we know, it has not yet been used for IP- or Ethernet traffic. In addition, the holistic schedulability analysis was developed for the sporadic model which is not a good match for MPEG encoded video-traffic. Another model, the generalized multiframe model [6] is designed to allow designers to express different sizes of video frames but unfortunately, it was not proposed for multihop communication; so far it has only been used to schedule a single resource.

In this paper, we present schedulability analysis of flows in multihop networks. Flows are characterized by the generalized multiframe model, the route of each flow is pre-specified and the output queue of each link schedules Ethernet frames by static-priority scheduling. We consider Ethernet switches implemented in software; this can be performed with Click [7], an open-source software package that implements basic functionalities of an Ethernet switch. We have used Click to implement an Ethernet switch with prioritized output queues, measured important characteristics of the implementation. The Click software uses stride scheduling [8] for scheduling software tasks inside the Ethernet switch. Hence those delays must be analyzed as well.

We consider the problem of satisfying real-time requirements from the perspective of a network operator who manages switches in the edge of the Internet and who is asked to offer delay guarantees to pre-specified flows. This requires that the network can identify which flow an incoming Ethernet frame belongs to; it can be solved but it is not the subject of this paper. As a network operator, we can only control the queuing discipline in the Ethernet switches — not the queuing discipline in the source node.

The remainder of this paper is organized as follows. Section 2 gives the necessary preliminaries. Section 3 presents the analysis of the response-time of a flow. Section 4 gives conclusions.

Preliminaries

We consider the problem of computing an upper bound on the response-time of a UDP packet in a multihop network comprising software-implemented Ethernet switches. The assumptions made and their relations to applications and our considered platform are given in this section.

Network model

Figure 1 depicts an example of the network considered. The network comprises nodes; some are Ethernet switches, some are IP-endhosts and some are IP-routers. On an IP-endhost there is one or many processes; each process is associated with one or many flows. For example, a process may be a video conferencing application and it may be associated with two flows: one for video and one for audio. A flow releases a (potentially infinite) sequence of UDP packets on the source node and these packets are relayed to the destination node by Ethernet switches.

The source node of a flow is either an IP-endhost or an IP-router. Analogously, the destination node of a flow is either an IP-endhost or an IP-router. The flow is associated with a route from the source to the destination; this route traverses only Ethernet switches — the route does not traverse IP-routers. Figure 2 shows an example of a route. Note that an IP-router may be a source node and then the destination node may be an IP-endhost; this happens if another node (outside the network we consider) sends data to the IP-endhost but we are only studying the Ethernet network and for this reason, the IP-router is the source node of the flow that is analyzed.

A flow releases a (potentially infinite) sequence of transmission requests where each transmission request means request to transmit a UDP packet. A packet could be for example an I-frame in an MPEG encoded video sequence. A UDP packet may be transmitted as a single Ethernet frame or it may be fragmented into several Ethernet frames. The Ethernet switches are not aware of the UDP packet; they are only aware of Ethernet frames. Despite this fact, we will describe the traffic over the Ethernet network using UDP packets and treat each UDP packet as a job in processor scheduling. Naturally this requires some adaptation. We will introduce a blocking term, and we will also need to introduce a new type of

Figure 2. Consider the network in Figure 1 and a flow with the source node being 0 and the destination node being 3. This figure shows the nodes forward packets of this flow. The arrivals of packets on node *0* are given by the generalized multiframe model.

jitter, called *generalized jitter* (explained in Section 2.3).

A transmitted Ethernet frame is received by another node. If this other node is the destination node of the flow then we say that the response time of the packet in the flow is the maximum time from when the UDP packet is enqueued at the source node until the UDP packet is received at the destination node of the flow. We say that the UDP packet is received at the destination node of the flow at the time when the destination node has received all Ethernet frames belonging to the UDP packet.

Figure 5 shows the internals of an Ethernet switch. If the node receiving an Ethernet frame is not the destination node of the flow then it is an Ethernet switch. The Ethernet switch receiving the Ethernet frame stores the Ethernet frame in a FIFO queue in the network card. The processor in the Ethernet switch dequeues the Ethernet frame from this FIFO queue and identifies the flow that the Ethernet frame belongs to. Based on this identification, the switch looks up in a table the outgoing network card that should be used and looks up the priority that the Ethernet frame should use. Each outgoing network interface has a corresponding priority queue, stored in main-memory. The Ethernet frame is enqueued into the proper outgoing queue. There is one software task for each ingoing network interface and this task performs this work. Each outgoing queue has a software task as well which checks if the FIFO queue of its corresponding network card is empty and if this is the case it dequeues an Ethernet frames from its corresponding priority queue and enqueues this Ethernet frame into the FIFO queue on the network card of the outgoing link. The network card naturally transmits the Ethernet frame on the link corresponding to the network card.

Let $link(N_1,N_2)$ denote the link between node N_1 to node N_2. $linkspeed(N_1,N_2)$ denotes the bitrate of $link(N_1,N_2)$. $prop(N_1,N_2)$ denotes the propagation delay (due to the finite speed of light) of $link(N_1,N_2)$.

Measurements on our implementation suggests that the uninterrupted execution time required for dequeuing an Ethernet frame from the incoming network card until it enqueues the Ethernet frame in the priority queue is 2.7µs. Measurements also suggests that the uninterrupted execution time required for dequeuing an Ethernet frame from the outgoing queue until it enqueues the Ethernet frame in the FIFO queue of the network card is 1.0µs. We assume that a single processor is used in the Ethernet switch and the processor is scheduled with stride scheduling.

Stride scheduling

Stride scheduling [8] is designed to (i) service tasks according to a pre-specified rate and (ii) have a low dispatching overhead. It works as follows. Each task is associated with a counter (called *pass*) and two static values: *tickets* and *stride*. The system also has a large integer constant. The *stride* of a task is this large integer divided by the *ticket* of a task. When the system boots, the *pass* (which is the counter) of a task is initialized to its *stride*. The dispatcher selects the task with the least *pass*; this task may execute until it finishes execution on the processor and then its *pass* is incremented by its *stride*. With this behavior, a task with *ticket*=2 will execute twice as frequently as a task with *ticket*=1. The amount of processing time used by the former task is not necessarily twice as much though.

Stride scheduling can be configured such that each task has a *ticket*=1; this causes stride scheduling to collapse to round-robin scheduling and we will use such configuration in the remainder of the paper[1].

Traffic model

As already mentioned, we assume the sequence of transmission requests is described with the generalized multiframe model. This model was originally developed for characterizing arrivals of jobs in processor scheduling but clearly it can be used for characterizing traffic in networks as well. The original generalized multiframe model did not model jitter. We will introduce jitter to this model but our notion of jitter is slightly different from the normal notion of jitter, we will call it *generalized jitter*.

A flow τ_i is a (potentially infinite) sequence of messages. Figure 3 gives an illustration of an MPEG stream. The MPEG stream requests to transmit UDP packets which are characterized by the generalized multiframe model. We are interested in finding the response time of a flow from source to destination. In order to do that, we will calculate the response time of the flow across a single resource (such as a link). And consequently, we need to describe how frequently the

[1] It is the default configuration in Click.

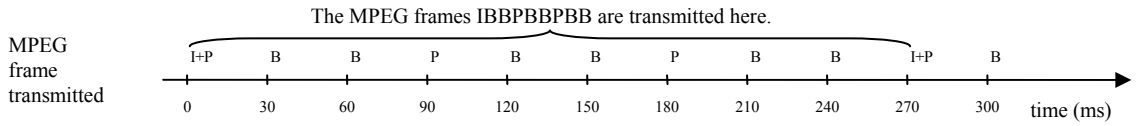


Figure 3. Consider a sequence of MPEG frames (=UDP packets), characterized as IBBPBBPBB and a movie which is comprised of a repetition of this sequence of frames. The P-frame stores the frame as a difference between the previous I- or P-frame. The B-frame stores the frame as a difference between the previous I-frame or P-frame or the next I-frame and P-frame. For this reason, the transmission order is as shown in the figure.

flow requests to use this resource and how much of the resource that it needs. The actual time needed depends on the characteristics of the resource, such as the link speed.

A flow τ_i is described with a tuple T_i, a tuple D_i, a tuple GJ_i, a tuple S_i and a scalar n_i. The scalar n_i represents the number of "frames" of the flow; these frames should not be confused with Ethernet frames. The flow for sending the MPEG stream given by Figure 3 has n_i=9 because there are 9 frames and then it repeats itself. The first frame is the UDP packet "I+P"; the second frame is the UDP packet "B" and so on.

Let $|T_i|$ denote the number of elements in the tuple T_i. Then it holds that $|T_i|=|D_i|=|GJ_i|=|S_i|=n_i$. The first element in the tuple T_i is indexed T_i^0 and it represents the minimum amount of time between the arrival of the first frame τ_i of and the second frame of τ_i at the source node. Analogously, for T_i^1, T_i^2,…,T_i^{ni-1}. Note that the exact times of transmission request of any frame is unknown; only lower bounds of inter-arrival times are known.

When a frame has arrived on the source node, it releases its Ethernet frames but all Ethernet frames are not necessarily released simultaneously. If t denotes the time when the first Ethernet frame of frame k of flow τ_i is released then all Ethernet frames of this frame are released during $[t,t+GJ_i^k)$. It can be seen that if all Ethernet frames of a frame would be released simultaneously and if Ethernet frames were arbitrarily small then our notion of jitter would be equivalent to the normal notion of jitter used in preemptive processor scheduling. Since GJ_i^k is a generalization, we say that GJ_i^k is the *generalized jitter* of frame k in flow τ_i.

The first element in the tuple D_i is indexed D_i^0 and it represents the relative deadline of the first frame; meaning that the first frame must reach the destination node within D_i^0 time units from the arrival on the source node. Analogously, for D_i^1, D_i^2,…,D_i^{ni-1}.

The first element in the tuple S_i is indexed S_i^0 and it represents the number of bits in the payload of the packet of the first frame. Analogously, for S_i^1, S_i^2,…,S_i^{ni-1}.

Schedulability Analysis

Basic parameters

We will now compute parameters for each link of each frame of a flow. By knowing the number of bits of payload in a UDP packet, it is possible to compute the transmission time of the UDP packet over a link with known link speed. A UDP packet must have an integral number of bytes and it must also include the UDP header (8 bytes). Let $nbits_i^k$ denote the number of bits that constitute the UDP frame (including the UDP header) of the k:th frame of flow τ_i. We have:

$$nbits_i^k = \left\lceil \frac{S_i^k}{8} \right\rceil \times 8 + 8*8$$

If Real-Time Transport Protocol (RTP) is used then it is necessary to add 16 bytes for the RTP header. Hence:

$$nbits_i^k = \left\lceil \frac{S_i^k}{8} \right\rceil \times 8 + 8 \times 8 + 16 \times 8$$

We must also add the IP-header (20 bytes). An Ethernet frame has a data payload of 1500 bytes and a header (14 bytes), CRC (4 bytes) and preamble+start-frame delimiter (8 bytes) and inter-frame gap (12 bytes). Therefore, an Ethernet frame has a maximum size of 12304 bits. Although the payload is 1500 bytes; 20 bytes of them are for the IP-header and hence there is room for 1480 bytes (=11840 bits) of data in each Ethernet frame. This gives us that $C_i^{k,link(s,d)}$, the transmission time of the UDP packet which is frame k of flow τ_i on *link*(s,d), can be computed as:

$$C_i^{k,link(s,d)} = \frac{\left\lfloor \frac{nbits_i^k}{11840} \right\rfloor \times 12304}{linkspeed\ (s,d)}$$

$$if \left\lfloor \frac{nbits_i^k}{11840} \right\rfloor \times 11840 \neq nbits_i^k \ then$$

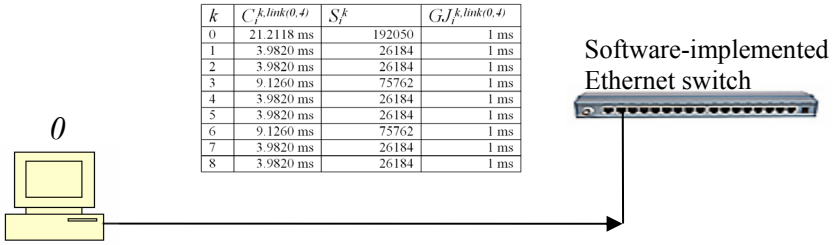
$$C_i^{k,link(s,d)} = C_i^{k,link(s,d)} + \frac{nbits_i^k - \left\lfloor \frac{nbits_i^k}{11840} \right\rfloor \times 11840 + 304}{linkspeed\ (s,d)}$$

$$end\ if$$

Let MFT (Maximum-Frame-Transmission-Time) be denoted as

$$MFT^{link(s,d)} = \frac{12304}{linkspeed(s,d)} \quad (1)$$

Let us consider the traffic in the MPEG stream in Figure 3 on the route given in Figure 2; call it flow τ_i. Consider the link from node 0 to node 4 and assume that *linkspeed(0,4)*=10^7 bit/s.



k	$C_i^{k,link(0,4)}$	S_i^k	$GJ_i^{k,link(0,4)}$
0	21.2118 ms	192050	1 ms
1	3.9820 ms	26184	1 ms
2	3.9820 ms	26184	1 ms
3	9.1260 ms	75762	1 ms
4	3.9820 ms	26184	1 ms
5	3.9820 ms	26184	1 ms
6	9.1260 ms	75762	1 ms
7	3.9820 ms	26184	1 ms
8	3.9820 ms	26184	1 ms

Figure 4. The parameters describing traffic over a specific link; here the link considered is *link*(0.4). This figure is a magnification of Figure 3, zooming in on the link from node 0 to node 4.

Calculations of $C_i^{k,link(0,4)}$ based on (1) and (2) yield the values shown in Figure 4. The parameters C_i^k for the other links *link*(4,6) and *link*(6,3) can be obtained analogously. Figure 3 showed the MPEG stream, assuming no generalized jitter. In practice there is generalized jitter; for the illustration in Figure 4 we assumed a generalized jitter of 1ms.

We will compute the response time of a frame k of a flow from source to destination; this requires that a pipeline of resources (each with a queue) is analyzed. We will compute the response time of the first resource and this becomes additional generalized jitter to the 2nd resource. We then compute the response time of the 2nd resource and so on by taking this generalized jitter into account. Finally, the response time from source to destination is obtained by adding the response times of all resources. If the response time from source to destination of every frame of a flow does not exceed its corresponding deadline then the flow meets all its deadlines.

The generalized jitter can be indexed in two different ways. GJ_i^k is the generalized jitter of the frame k of flow i of the source node; this is a specification of the flow. $GJ_i^{k,link(N1,N2)}$ represents the jitter of frame k of flow i on the link from node N_1 to N_2; this will be calculated.

In the analysis performed in this section, some short-hand notations are useful. $flows(N_1,N_2)$ denotes the set of flows over the link from node N_1 to node N_2. $hep(\tau_i, N_1, N_2)$ denotes the set of flows over the link from node N_1 to node N_2 which have higher priority than flow τ_i or equal priority as τ_i. $succ(\tau_j,N)$ denotes the node that is the successor of node N in the route of the flow τ_j. Analogously, $prec(\tau_j,N,)$ denotes the node that is the predecessor of node N in the route of the flow τ_j. $hep(\tau_j,N)$ and $lp(\tau_j,N)$ represent higher- and lower-priority flows, leaving node N. Formally they are expressed as:

$$hep(\tau_i, N) = \{j : (j \neq i) \wedge (j \in flows(N, succ(\tau_i, N))) \wedge (prio(j, N, succ(\tau_i, N)) \geq prio(i, N, succ(\tau_i, N)))\} \quad (2)$$

and

$$lp(\tau_i, N) = (flows(N, succ(\tau_i, N)) \setminus hep(i, N)) \setminus \{i\} \quad (3)$$

Further definitions follow below:

$$CSUM_j^{link(N_1,N_2)} = \sum_{k=0}^{n_j-1} C_j^{k,link(N_1,N_2)} \quad (4)$$

and

$$NSUM_j^{link(N_1,N_2)} = \sum_{k=0}^{n_j-1} \left\lceil \frac{C_j^{k,link(N_1,N_2)}}{MFT^{link(N_1,N_2)}} \right\rceil \quad (5)$$

and

$$TSUM_j = \sum_{k=0}^{n_j-1} T_j^k \quad (6)$$

Intuitively, (4) calculates the sum of the transmission times of all n_j frames of flow τ_j. Using the example, in Figure 4, we obtain:

$$CSUM_j^{link(N_1,N_2)} = 63.3628 \text{ ms}$$

Equation (5) calculates the number of Ethernet frames of all n_j frames of flow τ_j. Using the example, in Figure 4, we obtain:

$$NSUM_j^{link(N_1,N_2)} = 49$$

Equation (6) calculates a lower bound on the amount of time from when a frame of flow τ_j is requested until this frame is requested again. Using the example, in Figure 4, we obtain:

$$TSUM_j = 270 \text{ ms}$$

Later in the analysis, we need to consider a sequence of frames. Equations (7), (8) and (9) present such expressions based on (4), (5) and (6).

$$CSUM_j^{link(N_1,N_2)}(k_1,k_2) = \sum_{k=k_1}^{k_1+k_2-1} C_j^{k \bmod n_j, link(N_1,N_2)} \quad (7)$$

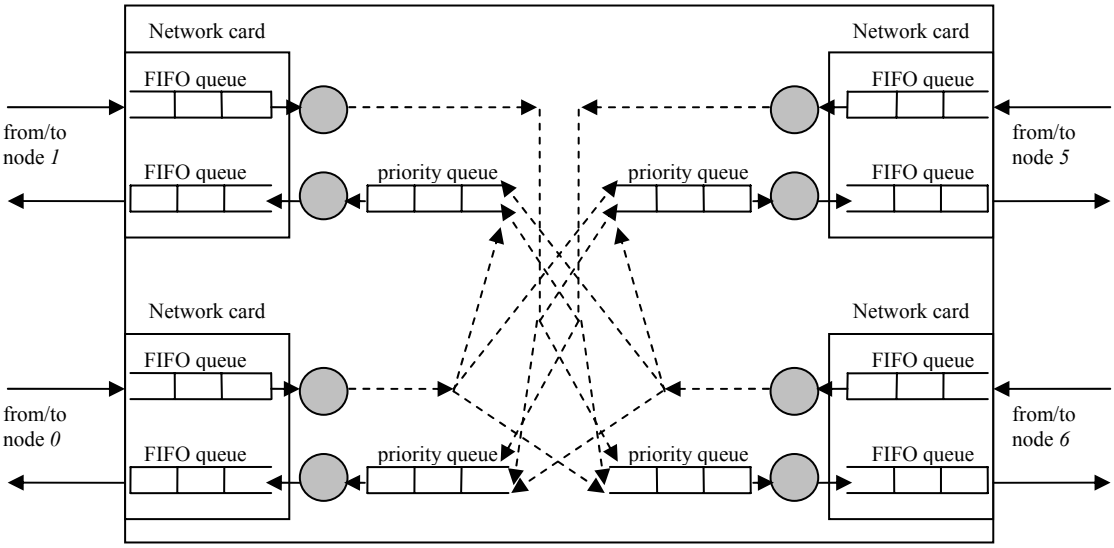


Figure 5. The internals of a software-implemented Ethernet switch. Arrows indicate the flow of Ethernet frames. A dashed line indicates the possibility of the flow of an Ethernet frame. A gray circle indicates a software task.

and

$$NSUM_j^{link(N_1,N_2)}(k_1,k_2) = \sum_{k=k_1}^{k_1+k_2-1} \left\lceil \frac{C_j^{k \bmod n_j, link(N_1,N_2)}}{MFT^{link(N_1,N_2)}} \right\rceil \quad (8)$$

and

$$TSUM_j(k_1,k_2) = \sum_{k=k_1}^{k_1+k_2-2} T_j^{k \bmod n_j} \quad (9)$$

Observe that the range of summation in (4),(5) and (6) are the same whereas the range of summation in (9) is different from the range of summation in (7) and (8).

$MXS(\tau_j,N_1,N_2,t)$ denotes an upper bound on the amount of time that flow τ_j uses the link from node N_1 to node N_2 during a time interval of length t. MXS is only defined for values of t such that $0<t<TSUM_j$. (S in MXS means small). The function MXS we use is:

$$MXS(\tau_j,N_1,N_2,t) = \min(t, \max_{k_1=0..n_j-1,k_2=1..n_j \text{ such that } TSUM_j(k_1,k_2)\le t} CSUM_j^{link(N_1,N_2)}(k_1,k_2)) \quad (10)$$

$MX(\tau_j,N_1,N_2,t)$ denotes an upper bound on the amount of time that flow τ_j uses the link from node N_1 to node N_2 during a time interval of length t. Unlike MXS, the function MX is defined for all positive values of t. The function MX we use is:

$$MX(\tau_j,N_1,N_2,t) = \left\lfloor \frac{t}{TSUM_j} \right\rfloor \times CSUM_j^{link(N_1,N_2)} + MXS\left(\tau_j,N_1,N_2,t - \left\lfloor \frac{t}{TSUM_j} \right\rfloor \times TSUM_j\right) \quad (11)$$

$NXS(\tau_j,N_1,N_2,t)$ denotes an upper bound on the number of Ethernet frames that are received from flow τ_j from the link from node N_1 to node N_2 during a time interval of length t. NXS is only defined for values of t such that $0<t<TSUM_j$. (S in NXS means small). The function NXS we use is:

$$NXS(\tau_j,N_1,N_2,t) = \max_{k_1=0..n_j-1,k_2=1..n_j \text{ such that } TSUM_j(k_1,k_2)\le t} NSUM_j^{link(N_1,N_2)}(k_1,k_2) \quad (12)$$

$NX(\tau_j,N_1,N_2,t)$ denotes an upper bound on the number of Ethernet frames that are received from flow τ_j from the link from node N_1 to node N_2 during a time interval of length t. Unlike NXS, the function NX is defined for all positive values of t. The function NX we use is:

$$NX(\tau_j,N_1,N_2,t) = \left\lfloor \frac{t}{TSUM_j} \right\rfloor \times NSUM_j + NXS\left(\tau_j,N_1,N_2,t - \left\lfloor \frac{t}{TSUM_j} \right\rfloor \times TSUM_j\right) \quad (13)$$

First hop

Recall that we consider the problem from the network operator´s perspective and hence we cannot make any assumption on the queuing discipline if the source node is an IP-endhost because the IP-endhost may be a normal PC running a non-real-time operating systems and has a queuing discipline in the network stack and queues in the network card that do not take deadlines into account. For this reason, we analyze the first hop assuming that Ethernet frames on the first link are scheduled by any

work-conserving queuing discipline. In our example network (in Figure 2), the first link is *link*(0,4).

Let $R_i^{k,link(S,succ(\tau i,S))}$ denote the response time of frame k in flow τ_i from the event that all Ethernet frames of frame k of flow τ_i has been enqueued on node S in the prioritized output queue towards node $succ(\tau_i,S)$ until all Ethernet frames of this frame have been received at node $succ(\tau_i,S)$. Let us define $extra_j(N,i)$ as:

$$extra_j(N,i) = \max_{k=0..nj-1} GJ_j^{k,link(N,succ(\tau_i,N))}$$

The method for computing R_i^k explores all messages released from flow τ_i during a so-called busy-period. The length of the busy period is computed as follows:

$$t_i^{k,link(S,succ(\tau_i,S)),0} = 0 \quad (14)$$

and iterate according to:

$$t_i^{k,link(S,succ(\tau_i,S)),v+1} = \sum_{j\in flows(S,succ(\tau_i,S))} MX\left(\tau_j, S, succ(\tau_i,S), t_i^{k,link(S,succ(\tau_i,S)),v} + extra_j(S,i)\right) \quad (15)$$

When (15) converges with $t_i^{k,link(S,succ(\tau i,S)),v+1} = t_i^{k,link(S,succ(\tau i,S)),v}$ then this is the value of $t_i^{k,link(S,succ(\tau i,S))}$. We can now compute $w_i^{k,link(S,succ(\tau i,S))}$ the queuing time of the q^{th} message of frame k in the busy period. It is computed iteratively until we obtain convergence, $w_i^{k,link(S,succ(\tau i,S)),v+1}(q) = w_i^{k,link(S,succ(\tau i,S)),v}(q)$ for the following iterative procedure:

$$w_i^{k,link(S,succ(\tau_i,S)),0}(q) = q \times CSUM_i^{link(S,succ(\tau_i,S))} \quad (16)$$

and iterate according to:

$$w_i^{k,link(S,succ(\tau_i,S)),v+1}(q) = q \times CSUM_i^{link(S,succ(\tau_i,S))} + \sum_{j\in flows(S,succ(\tau_i,S))\setminus\{i\}} MX\left(\tau_j, S, succ(\tau_i,S), w_i^{k,link(S,succ(\tau_i,S)),v}(q) + extra_j(S,i)\right) \quad (17)$$

when (17) converges with $w_i^{k,link(S,succ(\tau i,S)),v+1}(q) = w_i^{k,link(S,succ(\tau i,S)),v}(q)$ then this is the value of $w_i^{k,link(S,succ(\tau i,S))}(q)$. We compute the response-time for the q^{th} arrival of frame k of flow i in the busy period as:

$$R_i^{k,link(S,succ(\tau_i,S))}(q) = w_i^{k,link(S,succ(\tau_i,S))}(q) - q \times TSUM_i + C_i^k \quad (18)$$

This is used to calculate the response time:

$$R_i^{k,link(S,succ(\tau_i,S))} = \left(\max_{q=0..Q_i^k-1} R_i^{k,link(S,succ(\tau_i,S))}(q)\right) + prop(S,succ(\tau_i,S)) \quad (19)$$

where Q_i^k is defined as:

$$Q_i^k = \left\lceil \frac{t_i^{k,link(S,succ(\tau_i,S))}}{TSUM_i} \right\rceil$$

This analysis works for the case that

$$\sum_{j\in flows(S,succ(\tau_i,S))} \frac{CSUM_j^{link(S,succ(\tau_i,S))}}{TSUM_j} < 1 \quad (20)$$

From Reception to Enqueueing in Priority Queue

Figure 5 shows the internals of the Ethernet switch. As already mentioned the Click software schedules the tasks non-preemptively according to stride scheduling. We will now analyze it. Let *NINTERFACES*(*N*) denote the number of network interfaces on node *N*. (As an illustration, the switch in Figure 5 has *NINTERFACES*(*N*)=4.) Let *CROUTE*(*N*) denote the computation time on node *N* required to dequeue an Ethernet packet from an Ethernet card, find its priority and outgoing queue and enqueuing the Ethernet frame. Let *CSEND*(*N*) denote the computation time on node *N* required to dequeue an Ethernet frame from the priority queue and then enqueue it to the FIFO queue of the Ethernet card. Consequently, a task is serviced once every *NINTERFACES*(*N*) × (*CROUTE*(*N*)+*CSEND*(*N*)). We let *CIRC*(*N*) denote this quantity. In the example in Figure 5, we have that a task is serviced every 4*(2.7+1)µs; that is every 14.8 µs.

Let $R_i^{k,in(N)}$ denote the response time of frame k in flow τ_i from the event that the all Ethernet frames of frame k of flow τ_i have been received on node N until all Ethernet frames of this frame has been enqueued in the right priority queue in the Ethernet switch.

The method for computing $R_i^{k,in(N)}$ explores all messages released from flow τ_i during a so-called busy-period. The length of the busy period is computed as follows:

$$t_i^{k,in(N),0} = 0 \quad (21)$$

and iterate according to:

$$t_i^{k,in(N),v+1} = \sum_{j\in flows(prec(\tau_i,N),N)} NX\left(\tau_j, S, prec(\tau_i,N), t_i^{k,in(N),v} + extra_j(S,i)\right) \times CIRC(N) \quad (22)$$

When (22) converges with $t_i^{k,in(N),v+1} = t_i^{k,in(N),v}$ then this is the value of $t_i^{k,in(N)}$. We can now compute $w_i^{k,in(N)}$ as the queuing time of the q^{th} message of frame k in the level-i busy period. It is computed iteratively until we obtain convergence, $w_i^{k,in(N),v+1}(q) = w_i^{k,in(N),v}(q)$ for the following iterative procedure:

$$w_i^{k,in(N),0}(q) = q \times CIRC(N) \quad (23)$$

and iterate according to:

$$w_i^{k,in(N),v+1}(q) = q \times CIRC(N) + \sum_{j \in flows(prec(\tau_i,N),N)\setminus\{i\}} NX\left(\tau_j, S, prec(\tau_i,N), w_i^{k,in(N)),v}(q) + extra_j(S,i)\right) \times CIRC(N) \quad (24)$$

when (24) converges with $w_i^{k,in(N),v+1}(q) = w_i^{k,in(N),v}(q)$ then this is the value of $w_i^{k,in(N)}(q)$. We compute the response-time for the q^{th} arrival of frame k of flow i in the busy period as:

$$R_i^{k,\ln(N)}(q) = w_i^{k,in(N)}(q) - q \times TSUM_i + CIRC(N) \quad (25)$$

This is used to calculate the response time:

$$R_i^{k,in(N)} = \left(\max_{q=0..Q_i^k-1} R_i^{k,in(N)}(q)\right) \quad (26)$$

where Q_i^k is defined as:

$$Q_i^k = \left\lceil \frac{t_i^{k,in(N)}}{TSUM_i} \right\rceil \quad (27)$$

From Dequeueing of Priority Queue to Transmission

Consider Figure 5 again. We are interested in finding the time from when all Ethernet frames of the UDP packet is enqueued in the priority queue until all Ethernet frames of the UDP packet have been enqueued in the FIFO queue of the network card of the outgoing link. This time depends on the transmission times of priorities with higher priority and such analysis is well-explored in the research literature. This time depends also on the stride scheduling because it can happen that the outgoing link is idle but the task that dequeues an Ethernet frame is not executing and then the outgoing link remains idle although there may be an Ethernet frame in the outgoing queue. For this reason, equations are slightly different.

Let $R_i^{k,link(N,succ(\tau i,N))}$ denote the response time of frame k in flow τ_i from the event that the all Ethernet frames of frame k of flow τ_i has been enqueued on node N in the prioritized output queue towards node $succ(\tau_i,N)$ until all Ethernet frames of this frame has been received at node $succ(\tau_i,N)$.

The method for computing $R_i^{k,link(N,succ(\tau i,N))}$ explores all messages released from flow τ_i during a so-called level-i busy-period. The length of the level-i busy period is computed as follows:

$$t_i^{k,link(N,succ(\tau_j,N)),0} = MFT^{link(N,succ(\tau_i,N))} \quad (28)$$

and iterate according to:

$$t_i^{k,link(N,succ(\tau_i,N)),v+1} = MFT^{link(N,succ(\tau_i,N))} + \sum_{j \in hep(N,succ(\tau_i,N))} MX\left(\tau_j, N, succ(\tau_i,N), t_i^{k,link(N,succ(\tau_i,N)),v} + extra_j(N,i)\right) + \sum_{j \in hep(N,succ(\tau_i,N))} NX\left(\tau_j, N, succ(\tau_i,N), t_i^{k,link(N,succ(\tau_i,N)),v} + extra_j(N,i)\right) \times CIRC(N) \quad (29)$$

When (29) converges with $t_i^{k,link(N,succ(\tau i,N)),v+1} = t_i^{k,link(N,succ(\tau i,N)),v}$ then this is the value of $t_i^{k,link(N,succ(\tau i,N))}$. We can now compute $w_i^{k,link(N,succ(\tau i,N))}$ the queuing time of the q^{th} message of frame in the level-i busy period. It is computed iteratively until we obtain convergence, $w_i^{k,link(N,succ(\tau i,N)),v+1}(q) = w_i^{k,link(N,succ(\tau i,N)),v}(q)$ for the following iterative procedure:

$$w_i^{k,link(N,succ(\tau_i,N)),0}(q) = MFT^{link(N,succ(\tau_i,N))} + q \times CSUM_i^{link(N,succ(\tau_i,N))} \quad (30)$$

and iterate according to:

$$w_i^{k,link(N,succ(\tau_i,N)),v+1}(q) = MFT^{link(N,succ(\tau_i,N))} + q \times CSUM_i^{link(N,succ(\tau_i,N))} + \sum_{j \in hep(N,succ(\tau_i,N))\setminus\{i\}} MX\left(\tau_j, N, succ(\tau_i,N), w_i^{k,link(N,succ(\tau_i,N)),v}(q) + extra_j(N,i)\right) + \sum_{j \in hep(N,succ(\tau_i,N))\setminus\{i\}} NX\left(\tau_j, N, succ(\tau_i,N), w_i^{k,link(N,succ(\tau_i,N)),v}(q) + extra_j(N,i)\right) \times CIRC(N) \quad (31)$$

when (31) converges with $w_i^{k,link(N,succ(\tau i,N)),v+1}(q) = w_i^{k,link(N,succ(\tau i,N)),v}(q)$ then this is the value of $w_i^{k,link(N,succ(\tau i,N))}(q)$. We compute the response-time for the q^{th} arrival of frame k of flow i in the busy period as:

$$R_i^{k,link(N,succ(\tau_i,N))}(q) = w_i^{k,link(N,succ(\tau_i,N))}(q) - q \times TSUM_i + C_i^k \quad (32)$$

This is used to calculate the response time:

$$R_i^{k,link(N,succ(\tau_i,N))} = \left(\max_{q=0..Q_i^k-1} R_i^{k,link(N,succ(\tau_i,N))}(q)\right) + prop(S, succ(\tau_i,N)) \quad (33)$$

where Q_i^k is defined as:

$$Q_i^k = \left\lceil \frac{t_i^{k,link(N,succ(\tau_i,N))}}{TSUM_i} \right\rceil$$

This analysis will not converge if

$$\sum_{j \in hep(N,succ(\tau_i,N))\setminus\{i\}} \frac{CSUM_j^{link(N,succ(\tau_i,N))}}{TSUM_j} \geq 1 \quad (34)$$

```
1.    N1 := SOURCE(τi)
2.    N2 := succ(τi,N1)
3.    RSUM := GJi^k;  JSUM := GJi^k
4.    while N2≠DESTINATION(τi) do
5.      N3 := succ(τi,N2)
6.
7.      if  N1= SOURCE(τi) then
8.        GJi^k,link(N1,N2) := JSUM
9.        R := calculate Ri^k,link(N1,N2) from (19) based on S=N1
10.       RSUM := RSUM + R;  JSUM := JSUM + R
11.     end if
12.
13.     GJi^k,in(N2)  :=  JSUM
14.     R := calculate Ri^k,in(N2) from (26) based on N=N2
15.     RSUM := RSUM + R;  JSUM := JSUM + R
16.
17.     GJi^k,link(N2,N3) := JSUM
18.     R := calculate Ri^k,link(N2,N3) from (33) based on N=N2
19.     RSUM := RSUM + R;   JSUM := JSUM + R
20.
21.     N1 := N2
22.     N2 := N3
23.   end while
24.   Ri^k := RSUM
```

Figure 6. An algorithm for computing R_i^k, an upper bound on the response time of a frame k of flow τ_i from the source node of the flow to the destination of the flow.

This analysis may converge if

$$\sum_{j \in hep(N, succ(\tau_i, N)) \setminus \{i\}} \frac{CSUM_j^{link(N, succ(\tau_i, N))}}{TSUM_j} < 1 \qquad (35)$$

Putting it all together

Having these equations, we are now able to calculate the response-time from source to destination of a frame k from flow τ_i. Figure 6 shows an algorithm that computes this assuming that the generalized jitter of all links of all frames of other flows are known.

In practice, this assumption is usually false. One can however extend the ideas of holistic schedulability analysis [5] to the case where only the generalized jitter of source nodes are known. It works like this. Assume that the generalized jitter on the source nodes for each flow is what is specified and assume for every flow that the generalized jitter for links that are not from the source, is zero. Then calculate response times of each resource along the pipeline using the algorithm in Figure 6. Then let the generalized jitter of a resource be as calculated in the algorithm in Figure 6. Repeat the process of calculating the response times and updating generalized jitter until the jitter updating leads to the same jitter already assumed. Then the values of R_i^k output from the algorithm in Figure 6 can be compared to their deadlines. And this forms an admission controller.

Conclusions

Schedulability analysis of switched Ethernet is well-explored in the research literature of real-time computing with a focus on industrial settings rather than voice-over-IP and video-conferencing. For this reason, the analysis in the previous research literature did neither deal with multihop networks nor the generalized multiframe model.

We have taken this step and presented a schedulability analysis for traffic generated according to the generalized multiframe model in multihop networks comprising software-implemented Ethernet switches.

It can be seen that CIRC(N), the time required until a task is served again, heavily influences the delay. Hence, faster processors are clearly needed. An alternative approach is the use of multiprocessors; this is typically the case of today´s network processors. If m, the number of processors, is equally divisible by $NINTERFACES(N)$, one can assign $NINTERFACES(N)/m$ network interfaces to each processor. Clearly, if a network interface is assigned to a processor then both of the tasks that are assigned to this network interface are assigned to that processor aswell. In this way, quite large switches are implementable. For example, if a network processor comprises 16 processors and each of them have the same capability as the PC running Click, then a 48 port switch can be implemented with a $NCIRC(N)$=11.1μs. Such a switch can comfortably deal with links of speed 1 Gigabit/s.

Acknowledgements

This work was partially funded by the Portuguese Science and Technology Foundation (Fundação para Ciência e Tecnologia - FCT) and the ARTIST2 Network of Excellence on Embedded Systems Design.

References

[1] "Telefonkaos i Region Skåne," in *Svenska dagbladet*, 2007.

[2] J. Evans and C. Filsfils, "Deploying Diffserv at the Network Edge for Tight SLAs, Part 1," in *IEEE Internet Computing.*, vol. 8, 2004, pp. 61-65.

[3] B. Turner, "Why There's No Internet QoS and Likely Never Will Be," in *Internet Telephony Magazine*, vol. 10, 2007.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205," 1997.

[5] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, vol. 40, pp. 117 – 134, 1994.

[6] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," *Real-Time Systems*, vol. 17, pp. 5-22, 1999.

[7] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, pp. 263-297, 2000.

[8] C. A. Waldspurger and W. E. Weihl, "Stride Scheduling: Deterministic Proportional-Share Resource Management," MIT Laboratory for Computer Science June 1995.