Halmstad University Post-Print

Less pessimistic worst-case delay analysis for packet-switched networks

Mattias Weckstén and Magnus Jonsson

N.B.: When citing this work, cite the original article.

©2008 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Wecksten M, Jonsson M. Less pessimistic worst-case delay analysis for packet-switched networks. In: IEEE International Conference on Emerging Technologies and Factory Automation, 2008. ETFA 2008. Piscataway, N.J.: IEEE; 2008. p. 1213-1219.

DOI: <http://dx.doi.org/10.1109/ETFA.2008.4638556>

Copyright: IEEE

Post-Print available at: Halmstad University DiVA
<http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-2139>

Less Pessimistic Worst-Case Delay Analysis for Packet-Switched Networks

Mattias Weckstén
Centre for Research on Embedded Systems
P O Box 823
SE-301 18 Halmstad
mattias.wecksten@hh.se

Magnus Jonsson
Centre for Research on Embedded Systems
P O Box 823
SE-301 18 Halmstad
magnus.jonsson@hh.se

Abstract

The rapid growth of distributed real-time systems creates a need for cheap and available network solutions while still fulfilling the real-time requirements. In this paper we propose a method for less pessimistic delay analysis for packet switched first come first serve network, when knowing the intervals of possible message generation. Experiments show that the proposed method generates the expected results according to theoretical limitations of the experiment cases. The experiments also show that the proposed method could be practically used for non-trivial systems. Suggestions are given for future work on how to relax traffic requirements and how to cope with circular dependencies.

1. Introduction

As the size of real-time systems grows, from single board computers to blade systems and even comprising whole local area networks, the requirement for cheap and available real-time communication has increased rapidly. As commercially off the shelf (COTS) networking equipment usually have no or limited support for hard real-time guarantees there is a need for competent analysis tools.

In this paper we present a method for calculation of the packet delay in multi-hop switched networks when the packet offset is taken into account. The result is a tighter prediction of the delay compared to general methods used for packet delay analysis. Requiring a bounded packet offset might seem like a very strict requirement but when it comes to real-time systems determinism is often preferred over performance and flexibility. On the other hand, no intelligence is required in the network, just simple COTS, first come first serve (FCFS), store and forward, packet switches. The only traffic shaping that takes place is performed in the end nodes without any alterations of the network protocols.

Worst-case timing analysis of packet switched multi-hop networks have in many cases relied on traffic regulation, deadline scheduling or similar in intermediate

nodes in some form [1][2][3][4], or have focused on special cases of the traffic distribution [5]. However, in [6] the author presents a generic method for calculations of the delays in a multi-hop switched network where traffic shaping, in the form of earliest deadline first scheduling, is only allowed in the source nodes. The main drawback with this solution is that it is still somewhat pessimistic, taking into account situations that will not occur during runtime. Our suggested solution is to consider the packet offsets which limit the existence of packets to small time windows, which reduce the pessimism because of the increased knowledge of where and when a packet will exist. The argument for this solution is that the designer of a real-time system might prefer stability and predictability over flexibility and performance.

The rest of the paper is organized as follows: In Section 2 the assumptions and system definition for the current work are outlined and described. In Section 3 the algorithm for calculation of the packet delay is presented. In Section 4, the parameters for the experiments are defined, while the experiments and the results are presented in Section 5. In Section 6 the whole paper and its results are concluded. Finally in Section 7 a number of different suggestions for future work are presented.

2. Assumptions and system definition

The focus of this work is to perform packet delay calculus for real-time computer systems with multiple nodes connected by a packet switched network with predictable routing, for the case where the interval of possible packet generation is known for each packet. The network consists of simple store and forward, FCFS, output queued switches that allow predictable routing, e.g. source routing or static routing tables. It is also assumed that all the resource scheduling is performed off-line and that the only type of real-time control is of the release time for the packets from the nodes. The analysis assumes a single top priority traffic class, but this could easily be generalized to allow prioritized traffic and traffic classes of lower priority. Despite this

limitation of control the goal of this work is to derive the worst-case end-to-end delay for any packet entering the network (see Figure 1).

Figure 1. A packet enters the network from a source node at a known time. How large will the worst-case end-to-end delay be?

The system consists of N processor nodes, not necessarily of the same sort, connected by a switched network. The network is collision-free in the sense that only point-to-point links are used, and all bidirectional links are full-duplex links. The switches are output buffered, support store and forward and handle packets in FCFS order. The buffers in the switches are assumed to be large enough to avoid any buffer overflow for the traffic pattern applied. When passing a switch, s , there is an non-deterministic but bounded switch-dependent delay, X_s . The cabling is considered to consist of short cables of similar length with a propagation delay of p_i for link i . Since the switches are assumed to handle packets by store and forward, a transmission delay, $tr_{i,j}$, is introduced for each hop, equaling the time it takes for a packet of size j bits to leave the sending queue onto link i , where $tr_{i,MIN}$ is the delay for a packet of minimal size.

Any node in the system can be scheduled to communicate with any other node in the system to which there is a path. The communication between two nodes consists of packets with real-time constraints, with at least the earliest allowed start time (EST) for sending a packet to the destination specified. To be able to handle packets with real-time constraints it is required that the nodes in the system are clock synchronized down to a deviation for any clock to less than Δ from the global time T . This means that a packet sent from a late node to an early node will seem to arrive 2Δ later than if all nodes were perfectly synchronized. Because of node specific behavior, such as operating system task interference or cache or pipeline acceleration indeterminism, a packet scheduled for transmission at local time t will start its transmission in the frame t to $t+$, where $+$ is the node-specific worst-case delay for such delays. It is assumed that all packets leaving a node have been off-line scheduled, which results in that the packet order from each node is known and preserved. In other words, if a packet is ready to be sent out from the node earlier than scheduled, transmission will still be delayed to the release time.

Although the end nodes are synchronized within a known factor, Δ , and all traffic has been off line scheduled, the packets are sent without any form of synchronization or traffic shaping when they once have

entered the network. This leads to a queuing delay, c_p , as a packet is passing a switch output queue out onto link i . However, because of the restriction on how packets are allowed to be released from a node (not ahead of time and in non overlapping order) the queuing delay for packets entering the outgoing link of a node will be 0. An overview of the delay components for a packet w of size $|w|$ bits, going from node $N1$ to node $N2$ via the switch $S1$, is seen in Figure 2.

Figure 2. Example of delay components affecting a packet.

3. Algorithm

To be able to calculate a bounded worst-case delay for a packet being sent over a network it is essential that all interference leads to a bounded delay. This is the case in the fully packet-switched networks assumed. The reason for this is that all communications are buffered in the sense that no two nodes are competing for the same link resource at any time, but rather queued in order to access the link sequentially. The queuing delay will be possible to calculate and will depend on interference from packets from other links sharing the same output buffer. However, the delay that could be introduced when passing a switch still depends on possible interference from packets on other links and the chosen packet queuing order. As stated earlier, in this paper the queuing order is assumed to be FCFS.

To be able to calculate the queuing delay for a certain packet w in the buffer B_p connected to switch s via the link i , we have to know the earliest starting time (EST) and latest finishing time (LFT) for all possibly interfering packets as this will affect the total delay. To calculate the EST for next hop, EST' , for packet w as it enters the link i , we just add the minimal possible delay for the hop to the EST . The minimal per hop delay for a packet is the case where no packet queuing delay occurs and thus equals the sum of the switch dependent delay, X_s , the propagation delay, p_i , and the minimal transmission delay, $tr_{i,MIN}$ (see Equation 1).

$$EST' = EST + X_s + p_i + tr_{i,MIN} \quad (1)$$

To calculate the LFT for the next hop, LFT' , the worst-case delay has to be calculated. Besides the switch dependent delay, X_s and the propagation delay, p_i , the worst-case transmission delay, $tr_{i,|w|}$, where $|w|$ is the

packet size in bits, has to be taken into account along with the queuing delay, c_i , caused by interference with other packets competing for the same buffer (see Equation 2).

$$LFT' = LFT + X_s + p_i + tr_{i|w|} + c_i \quad (2)$$

To be able to compete for the buffer the competing packets must be able to enter the queue at the LFT for packet w at latest. This leads to the fact that the LFT' for a packet w in buffer B_i depends on all packets destined for the buffer B_i with an EST that is less or equal to the LFT of w .

To calculate the LFT' for packet w , LFT'_w , we need to know the worst-case buffer load at the LFT of w , LFT_w , since this is the latest time at which packet w will be queued and thus no longer affected by interfering packets after that. The queuing delay c_i will equal the transmission delay for all data in the buffer B_i at LFT_w .

To calculate the buffer state there is actually no need to calculate the contribution of each and every packet individually, the flows from each input could be generalized to worst-case data flows called work loads. To generate a workload for a packet a from a link i to a buffer B_j , we simply assume that the packet will be sent as tight as possible towards the LFT_w . A workload for a single packet a , from the link i , with the EST equal to EST_a , a transmission delay of $tr_{i|a|}$ and a LFT equal to LFT_a have three possible outcomes: If $LFT_a < LFT_w$ then the workload will range from $LFT_a - tr_{i|a|}$ to LFT_a , see Figure 3.

Figure 3. $LFT_a < LFT_w$

If $EST_a + tr_{i|a|} \leq LFT_w \leq LFT_a$ then the workload will range from $LFT_w - tr_{i|a|}$ to LFT_w , see Figure 4.

Figure 4. $EST_a + tr_{i|a|} \leq LFT_w \leq LFT_a$

And if $EST_a + tr_{i,MIN} < LFT_w < EST_a + tr_{i|a|}$ then the workload will range from EST_a to LFT_w , see Figure 5.

To generate the workload for a whole incoming link we have to consider two cases, either the packet workloads are overlapping or they are not overlapping.

In the case where a packet workload has no overlap the workload is simply the sum of the packet workloads. If two or more packet workloads are overlapping the result will be a single larger workload. For example, consider the two packets a and b that have overlapping packet workloads; the resulting workload will range from $MAX(LFT_a, LFT_b) - tr_{i|a|} - tr_{i|b|}$ to $MAX(LFT_a, LFT_b)$, see Figure 6.

Figure 5. $EST_a < LFT_w < EST_a + tr_{i|a|}$

Figure 6. Overlapping packet workloads.

With all link workloads calculated it is now possible to calculate the buffer state. This is done by adding all workloads at incoming links and operating a smoothing operator over the resulting function to simulate the buffer behavior, that is; consume one unit data for each time unit if data is available. When LFT_w is reached the buffer could be empty which means that the queuing delay was zero or the buffer could hold a number of data units that must be consumed from the buffer before the packet w will be sent. The transmission delay for the amount of data left in the buffer will equal the queuing delay, see Figure 7.

Figure 7. The workload functions are added and smoothed out.

On a side note; a possible practical problem when designing the algorithm is that an evaluation of the delays for each possible point of time between 0 and LFT_w will lead to an algorithm complexity dependent on the timing granularity. However, there is no need to evaluate the delay functions in all possible points of time; it is sufficient to check the points where any change of the underlying parameters are made. In practice this means that the number of evaluation points for each packet w is approximately the number of packets with an EST less or equal to LFT_w .

4. Experiment set-up

To be able to use the algorithm described in the previous section to calculate individual packet delays, one of the requirements is that the delays for all packets the calculation depends on must be previously calculated. The feasibility of that task is strongly dependent on the traffic pattern, topology and the routing algorithm. In this paper we will make assumptions about the topology and routing algorithm that simplifies the calculations, while a further discussion on the general case will be found in the future work section.

Figure 8. The used internal connections of a switch in a shortest path routed binary tree.

The topology chosen for the experiments is a binary tree, with the computation nodes in the leaves and switch nodes in the rest of the tree. This is done for clarity reasons; any tree topology could be used without any modification of the algorithm. Shortest path routing is used. This combination leads to a number of interesting properties. The shortest path in a binary tree (or any non-fat tree actually) results in a unique path for any source, destination pair. It is also possible to generate a relatively simple calculation order for the delay analysis of the packets that guarantees that all dependencies (LFT s and EST s for earlier steps) are solved. When analyzing how the shortest path algorithm route packets over the topology it is evident that the buffers in the switches of the topology will be of one of two types, either an upward buffer or a downward buffer (see Figure 8). The delay analysis of packets in the upward buffers will only depend on packets from lower layers in the tree (see Figure 9) while packets in the downward

buffers depend on both packets from upward buffers and downward buffers in the tree (see Figure 10).

Our method to guarantee that there are no unsolved dependencies at any time is to start the calculations in the lowest layer and calculate delays for all upward buffers. When no more upward buffers are available the downward buffers are treated, starting at the root (top) node and working downwards through the layers.

Figure 9. Packets in the upward buffers only depend on packets from the lower levels.

Figure 10. With all upward traffic solved, the downward traffic will only have unsolved dependencies in higher levels.

Besides topology and routing algorithm the operating system latencies, clock drift, cable delay, link data rate and internal switch delay have to be determined. Typical real-time operating systems have latencies in the range of microseconds to tens of microseconds. For this experiment it is assumed that the worst-case operating system delay is 10 microseconds, which is realistic [7]. The nodes are assumed to be clock synchronized but in practice there will be a clock drift in between synchronizations. However, typical clock drift in a clock synchronized wide area real-time system range from hundred nanoseconds up to microseconds [8], so in this experiment it is assumed that the clock drift is negligible. It is assumed that all cables in the system are short, less than a meter, and of approximately same length which results in a cable delay in the nanosecond range, which is negligible. The internal switch delay is the worst-case delay for a packet passing through a switch excluding the queuing delay. This factor is in the micro second range and assumed to be identical for all switches and is set to 1 microsecond. Since it is assumed that store and

forward switches are used there are no restrictions on allowing links of different data rate (it is supported by the algorithm as well) but to keep things simple in the experiment set-up all links in the system will have a data rate of 100 Mbit/s. For comparison, the duration of a packet with the selected data rate of 100 Mbit/s range from approximately ten microseconds up to approximately 100 microseconds. In our experiments we simulate balanced binary tree topologies with $n=2^N$ leaf nodes and thus $2^{N+1}-1$ nodes. This way the number of leaf nodes uniquely describes the topology.

Traffic for the experiment should be randomly generated, meeting the requirements for the packets; fixed transmission order and that packets from the same source does not overlap. The parameters of a packet are source node, destination node, packet size, *EST* and *LFT*. To generate these parameters for a certain traffic intensity, *TI*, while guaranteeing the requirements, the following algorithm is used:

1. Randomly generate a source node, s , in the range $[0, n-1]$
2. Randomly generate a destination node, $d \neq s$
 - a. Randomly generate a destination offset, o_d , in the range $[1, n-1]$
 - b. Calculate the destination d as the wrap around sum of the source and the offset, $((s + o_d) \% n)$
3. Randomly generate a packet size, p , in the range $[512, 12000]$ bits
4. Randomly generate an *EST*
 - a. Get a time offset, o_p , by retrieving the *LFT* for the last packet generated at node s , or 0 if this is the first packet
 - b. Randomly generate a packet to packet time interspace, g , in ms with the expected value $62.56 \cdot ((1/TI)-1)$
 - c. Calculate the *EST* as $g + o_p$
 - d. Calculate the *LFT* as the sum of *EST*, the operating system delay, ϕ , and the transmission delay for a packet of size p .

This is repeated until there are packets with a *LFT* larger than the experiment length for all source nodes. The randomization distributions could of course be of any type, but in our case we decided on rectangular distributions.

5. Experiments

Given the experiment setup outlined in previous section, a number of experiments were performed for topologies with 4, 8, 16, 32 and 64 leaf nodes. Random packets were generated at a given traffic intensity and transmitted over the network. Above some level of traffic intensity the buffers in the hot spots will receive more data on average than they are able to transmit, thus the buffers will saturate. This will show very clearly

when analyzing the worst-case packet delay, since the buffers are saturated, the delay will increase rapidly. To calculate this point we formulate the function for the average traffic intensity *TOP* through the top node of a balanced binary tree topology. According to previous section the destination for any packet is evenly distributed which means that with n end-nodes in total there will be $n/2$ possible destinations where the packet will pass through the top node. The probability that a packet will pass through the top node then equals to the number of positive outcomes ($n/2$) divided by the number of possible outcomes that is $n-1$. The traffic intensity at each source node is *TI* and saturation occurs when *TOP* is equal to 1 (see Equations 6 and 7).

$$TOP = TI \cdot \frac{n/2}{n-1} \cdot \frac{n}{2} = 1 \quad (6)$$

$$TI = \frac{n-1}{n^2/4} \quad (7)$$

Evaluated for the topologies included in the experiments it gives the following result:

Leaf nodes	Sat. level
4	75%
8	44%
16	23%
32	12%
64	6%

Table 1. Saturation levels for different number of nodes in the topology.

If an experiment is repeated with the same parameters except for the length of the simulation the resulting worst-case delays will be close to the same in average for traffic intensities lower than the saturation level for the topology. However, for traffic intensities greater than the saturation level the longer experiment will show a greater increase in averaged worst-case delay, since there are saturated buffers. We decided to double the experiment length for each parameter set and plot the two curves side by side for comparison, resulting in experiment lengths of 10 ms and 20 ms respectively. For traffic intensities below the saturation level the both curves should keep close together and above the saturation level they should be spread apart. The experiments were repeated 100 times for each parameter setting and the average of the worst-case delay was plotted against the traffic intensity.

In Figures 11-15 it is seen that the delays start increasing before the calculated saturation points, which was expected behavior. For the experiment in Figure 11 the delay seems to increase linearly until approximately a traffic intensity of 60-70%, after which the delay turns into an exponential growth. In Figure 12 the change seems to come at a traffic intensity of 35%, in Figure 13

Figure 11. The experiments for topologies with 4 nodes show a clear increase in delay at the saturation level (75%).

Figure 13. The experiments for topologies with 16 nodes show a clear increase in delay at the saturation level (23%).

Figure 12. The experiments for topologies with 8 nodes show a clear increase in delay at the saturation level (44%).

Figure 14. The experiments for topologies with 32 nodes show a clear increase in delay at the saturation level (12%).

at a traffic intensity of 20%, in Figure 14 at a traffic intensity of 10-11%, and in Figure 15 at a traffic intensity of 5-6%.

6. Conclusions

The aim of this work has been to take advantage of the knowledge about packet offsets in the end nodes to generate tighter end-to-end delay estimates for packet switched traffic in COTS switched networks.

When experimentally evaluated, our algorithm shows results that behave as expected from the theoretical limitations for the type of networks that were used. It has also been shown that the algorithm is applicable for non-trivial cases consisting of up to 64 end-nodes with intensive traffic patterns.

Although the focus for this paper was to analyze the delay for single packets it is also possible to analyze the worst-case buffer utilization or for how long the system must remain silent to recover after a burst.

7. Future Work

The basis of this work was to formulate an algorithm that made it possible to calculate tight or non pessimistic boundaries on packet delays in switched networks. As

Figure 15. The experiments for topologies with 64 nodes show a clear increase in delay at the saturation level (6%).

shown in earlier sections this is possible in a deterministic number of steps. However, some of the requirements assumed in this paper might seem strict. In this section we will explain how some of these requirements could be relaxed.

One of the requirements on the end nodes was that all traffic was off-line scheduled and that packets are of a known order and non-overlapping. This requirement was added to get a starting point in the calculations since each packet must have an EST and a LFT. However, as long as the packets on the first link have their EST and a

LFT calculated neither order nor overlap avoidance is needed. To still be able to calculate these parameters we have to model the scheduling algorithm used in the end nodes. If it is assumed that the scheduler is a simple FCFS scheduler, it is the case that this could be modeled using one of the switch nodes, see Figure 16. The parameters needed for the scheduler would be the release time (i.e. EST) and the deadline for sending (i.e. LST).

Figure 16. By introducing a virtual switch, the exact packet order does not have to be known.

The restriction in using store and forward switches could easily be relaxed since this is only affecting the timing variables that are added in each hop between the switches. So, with minor changes the algorithm will support cut through routing as well. Similarly for the choice of FCFS queuing, as long as interference is limited to the live span of the packet, any reasonable strategy should be possible to adapt to.

Another restriction introduced in this paper is the topology and routing restriction. This was introduced to make it very simple and straight forward to find the calculation order and avoiding circular dependencies. Instead of creating a situation where a calculation order is given it is possible to create a function that checks the number of unsolved dependencies a certain packet in a certain buffer has for the moment. This makes it possible to sort out those packets that do not have any unsolved dependencies and calculate their LFT. However, if the topology or routing requirements are relaxed this could result in circular dependency chains which prevent the hierarchical solving. The suggested solution for this is to break detected circular dependencies by introducing a pessimistic estimate for one of the packets in the dependency chain. The choice of the estimate could simply be the case where all unknown packets with dependencies are assumed to generate worst possible blocking. This is possible since, although the exact live

span might be unknown, the worst-case workload of the packet is known, which makes it possible to calculate the LFT iteratively. When the maximum change in LFT of the packets in the chain is below a certain factor the iterations are stopped. Given that the initial estimate is truly pessimistic the results down the dependency chain will never be optimistic since a pessimistic input always generates a pessimistic output. It might be noted that if the difference in LFT for any packet reaches 0 this means that a stable solution has been reached. As an alternative there are other mildly restrictive solutions for avoidance of dependency chains, for example x-y routing or any other deadlock free protocol.

References

- [1] Ferrari, D.; Verma, D.C., "A scheme for real-time channel establishment in wide-area networks," *Selected Areas in Communications, IEEE Journal on*, vol.8, no.3, pp.368-379, Apr 1990
- [2] Palencia, J.C.; Harbour, M.G., "Response time analysis of EDF distributed real-time systems," *Journal of Embedded Computing*, vol.1, no.2, pp. 225-237, 2005
- [3] Qin Zheng; Shin, K.G., "On the ability of establishing real-time channels in point-to-point packet-switched networks," *Communications, IEEE Transactions on*, vol.42, no.234, pp.1096-1105, Feb/Mar/Apr 1994
- [4] Kandlur, D.D.; Shin, K.G.; Ferrari, D., "Real-time communication in multi-hop networks," *Distributed Computing Systems, 1991., 11th International Conference on*, pp.300-307, 20-24 May 1991
- [5] Lee, K.C.; Lee, S.; Lee, M.H., "Worst Case Communication Delay of Real-Time Industrial Switched Ethernet With Multiple Levels," *Industrial Electronics, IEEE Transactions on*, vol.53, no.5, pp.1669-1676, Oct. 2006
- [6] Fan, X., *Real-Time Services in Packet-Switched Networks for Embedded Applications*. Göteborg: Chalmers University of Technology. Doctoral thesis, 2007
- [7] Victor Yodaiken et al., "RTLinux/RTCore dual kernel real-time operating system", FSMLabs, White Paper, Mar 2003
- [8] Schossmaier, K.; Loy, D., "An ASIC supporting external clock synchronization for distributed real-time systems," *Real-Time Systems, 1996., Proceedings of the Eighth Euromicro Workshop on*, pp.277-282, 12-14 June 1996