Fail-Aware Publish/Subscribe

Zbigniew Jerzak, Robert Fach, Christof Fetzer
Dresden University of Technology
01062 Dresden, Germany
{Zbigniew.Jerzak, Robert.Fach, Christof.Fetzer}@*tu-dresden.de*

Abstract

In this paper we present a wide area distributed system using a content-based publish/subscribe communication middleware which can deterministically detect and report failures with respect to timely message delivery. Our approach does not require external clock synchronization nor does it impose any constraints on the publish/subscribe middleware. We show that our system performs better and is safer than when using NTP for external clock synchronization.

1. Introduction and Motivation

For many critical applications, e.g., Critical Infrastructure Protection [24] (CIP) it is essential to be aware whether one: (1) has received all information and (2) the information that was received is timely.

The critical applications include, among others, transportation [8], electric grid related [25] and embedded [13] system. Critical applications and systems are typically geographically distributed and thus require a Wide Area Network (WAN) to collect and manage information. Even in areas where WAN use has been so far limited (e.g. electric grid control), the demand for its introduction is clearly increasing [12]. Providing timeliness guarantees for WANs is thus of vital importance for the safety and the wide adoption of such systems.

Wide area networks simultaneously pose a challenging environment often suffering from failures, including delays and information loss. One reason for this are network partitions and routing anomalies – instances where live nodes are not able to route packets to each other. Measurements have shown that a large wide area distributed system (PlanetLab) consisting of 280 nodes partitions at least once a day. It has been shown that such partitions could last up to days as they needed to be either fixed manually or the partitioned nodes needed to be restarted. Moreover, within a ten day period, at least one node suffers from a routing anomaly [17].

In this paper, we present a wide area publish/subscribe middleware which permits to compute upper bounds on the transmission delay and hence to determine the timeliness of the received information. Experiments carried out in the PlanetLab [1] environment show that proposed approach is a viable alternative to external clock synchronization using NTP [15, 16]. We also show that external clock synchronization using NTP does not allow for the construction of the fail-aware systems.

2. Related Work

Publish/Subscribe systems span from topic-based [5] to content-based [7] solutions which are deployed using different architectures ranging from centralized topologies [22], through acyclic networks of servers [3] to peer-to-peer [23]. We have decided to base our work on one of the standard content-based publish/subscribe systems – Siena [3].

Despite the broad range of publish/subscribe systems only a few provide guarantees related to the delivery of messages or timeliness in a wide area networks environment. The approach presented in [26] allows for in order, gapless delivery of information. However, it does not make any assumptions as to the timeliness of messages. [18] presents a partition-aware system. However, its implementation relies on the externally synchronized clocks and is not used at the publish/subscribe level. [9] augments a topic-based publish/subscribe system with the ability to detect if all messages that are at least some D seconds old where received. Our work in contrast uses a content-based publish/subscribe system and alleviates the requirement for the externally synchronized clocks.

On the other hand, there exist systems providing real-time properties [21], however, they require dedi-

cated hardware and use closed source software, so it is neither possible to evaluate their functioning nor feasible to use them with the existing infrastructure (e.g., the Internet). Similarly, [12] requires the use of dedicated infrastructure and externally synchronized clocks.

3. Fail-Awareness

Fail-awareness provides an indicator allowing to implement services in distributed systems with uncertain communication and access to local hardware clocks. The indicator tells whether some safety property currently holds or if it might be violated [10]. We present the assumptions for our fail-aware system which are based on the Timed Asynchronous Distributed System Model [6]: (1) all processes are timed, (2) processes have crash/performance failure semantics, (3) all processes have access to a local, unsynchronized hardware clock with a bounded drift rate and (4) there exists no upper bound on the communication frequency nor the number of failures in the system.

Intuitively, a fail-aware system is a system which is able to detect when it is not possible to depend upon properties provided by lower level services, e.g, to depend upon the provided information because its transmission time violates the predefined Δ threshold [11]. The Timed Asynchronous Distributed System Model is suitably weak to represent the contemporary distributed systems. Assumptions it makes on the type and frequency of failures, communication infrastructure and time flow allow it to be implemented in existing systems without the need to modify them.

4. Upper Bound on Transmission Delay

A message m is *late* if its transmission time is greater than a predefined value δ. It is *timely* otherwise. It is only possible to detect a message that is late if it is possible to calculate the transmission time of that message. However, it is not possible to calculate the transmission time by subtracting send and receive time stamps in a system with unsynchronized clocks.

Let us assume that it is possible to calculate an upper bound $ub(m)$ on the transmission delay $td(m)$ of a message m, i.e., it can be guaranteed that the transmission time is not greater than the calculated value: $td(m) \leq ub(m)$.

Whenever the calculated upper bound value $ub(m)$ is less or equal to some given Δ, we say that the message is *fast*. When $ub(m) > \Delta$, we say that the message is *slow*. Typically, slow messages might need to be rejected by an application and Δ will be application and sometimes even message specific. However, we abstract from this by using only slow and fast message indicators. To compute an upper bound $ub(m)$ on a transmission delay of the message m between two connected processes p and q (see Figure 1) each of them requires a monotonic local clock with a bounded drift rate ρ with respect to real time – see Section 3. A drift rate of process q is bounded by ρ_q and the drift rate of process p is bounded by ρ_p.

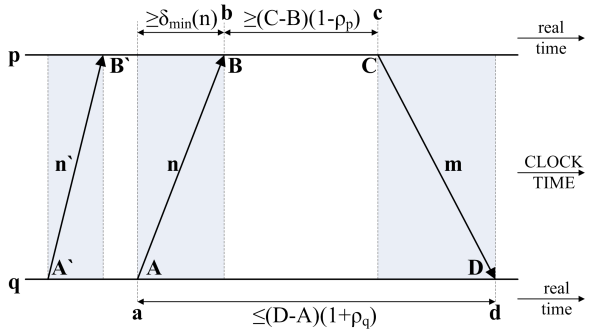


Figure 1. Calculating an upper bound on the transmission delay $td(m) = d - c$.

The intuition behind the method is that process q sends a helper message n to process p at time A – measured with its local clock. Subsequently, p sends a message m to q which receives it at time D indicated by q's local clock. The transmission delay $td(m)$ of message m is bounded by the real-time period between D and A. The error due to q's clock drift is bounded by $(D-A)\rho_q$. Hence, $td(m) \leq (D-A)(1+\rho_q)$.

The upper bound calculation can be improved by subtracting the processing time at process p. Hence $td(m) \leq (D-A)(1+\rho_q) - (C-B)(1-\rho_p)$.

A further improvement can be achieved if we are able to calculate a lower bound $lb(n)$ on the transmission delay of the helper message n, i.e., $lb(m) \leq td(m)$. Knowing the bandwidth of the network ($bndwdth$) connecting p and q and the size ($size$) of the helper message n, a lower bound on the transmission delay of the helper message n can be set to $\delta_{min}(n) = \frac{size}{bndwdth}$. Hence, we define the upper bound $ub(m)$ of a message m as:

$$ub(m) = (D-A)(1+\rho_q) - (C-B)(1-\rho_p) - \delta_{min}(n) \quad (1)$$

We define an error $e_{ub}(m)$ on the upper bound $ub(m)$ of the transmission delay $td(m)$ of a message m as the difference between the calculated upper bound and the real time transmission delay, i.e.: $e_{ub} = ub(m) - td(m)$ (see Figure 1). We can see from Equation 1 that with an increase of the transmission delay $td(n)$ (more precisely, with an increase of the uncertainty $td(n) - \delta_{min}(n)$) so will the $ub(m)$ and the error value $e_{ub}(m)$. To cope with this problem, we need a helper message n with a smaller uncertainty $td(n) - \delta_{min}(n)$ – see Figure 1. Instead of using more recent helper n, we can use n' with a small

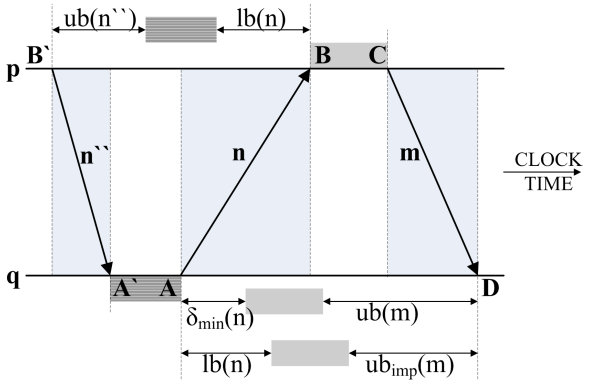


Figure 2. An improved upper bound. When the lower bound $lb(n)$ of helper message n is greater than $\delta_{min}(n)$, the improved upper bound $ub_{imp}(m)$ is smaller than the original $ub(m)$.

uncertainty. The criterion for choosing n' in favor of n can be expressed as:

$$[(D-A')(1+\rho_q)-(C-B')(1-\rho_p)-\delta_{min}(n')] < [(D-A)(1+\rho_q)-(C-B)(1-\rho_p)-\delta_{min}(n)]$$

Which results in:

$$(A-A')(1+\rho_q)-\delta_{min}(n') < (B-B')(1-\rho_p)-\delta_{min}(n) \quad (2)$$

[4] proposes to calculate a lower bound which can be greater than $\delta_{min}(n)$. We show how to use this improved lower bound to improve the upper bound.

Let us consider a pair of messages n'' and n (see Figure 2 – for presentation clarity we assume that drift rate ρ is equal to zero). We assume that $lb(n)$ represents the lower bound on the transmission delay of message n. It can be calculated as:

$$lb(n) = (B-B')(1-\rho_p)-(A-A')(1+\rho_q) - ub(n'') \quad (3)$$

It is possible that the calculated lower bound $lb(n)$ is smaller than the $\delta_{min}(n)$. Specifically, it can be less than zero. Therefore, we define an improved lower bound $lb_{imp}(n)$ as:

$$lb_{imp}(n) = \begin{cases} \delta_{min}(n) & \text{if } \delta_{min}(n) >= lb(n) \\ lb(n) & \text{if } \delta_{min}(n) < lb(n) \end{cases} \quad (4)$$

Process p can subsequently attach the calculated value $lb_{imp}(n)$ on message m that it sends to process q (see Figure 2). Process q knowing the value of $lb_{imp}(n)$ which it has received with m, can calculate an improved upper bound $ub_{imp}(m)$:

$$procT = (D-A)(1+\rho_q)-(C-B)(1-\rho_p)$$
$$ub_{imp}(m) = procT - lb_{imp}(n) \quad (5)$$

The ability to calculate the lower bound on a helper message influences the choice of the faster helper message presented in Equation 2:

$$(C-C')(1+\rho_q)-lb_{imp}(n') < (D-D')(1-\rho_p)-lb_{imp}(n) \quad (6)$$

Where $lb_{imp}(n')$ and $lb_{imp}(n)$ are the improved lower bounds on helper messages n' and n (see Figure 1).

5. Content-Based Publish/Subscribe

Publish/Subscribe (pub/sub) is a communication paradigm which allows for (1) asynchronous and (2) decoupled communication between information sources and information sinks. Asynchronous communication implies that information sources and sinks do not block when communicating with each other. Information sources are called publishers and information sinks are called subscribers. The decoupled communication can be characterized as follows: (1) communication between publishers and subscribers is anonymous and (2) communication does not need to take place at the same time.

Content-based is one of most expressive types of a pub/sub systems. Information in content-based pub/sub needs not to be ordered into predefined topics, rather it is freely expressed and matched upon. In *S*IENA there are three types of participants: (1) subscribers, (2) publishers and (3) routers.

In order to receive information subscribers issue subscriptions which are broadcasted into the network. Subsequently, publishers issue publications which are actual pieces of information. A router, upon receiving a publication, compares its content against stored subscriptions‘ content (content-based matching) and whenever it encounters a match (e.g., subscription $x < 5; y > 3$ and publication $x = 1; y = 5$) it forwards the publication on the reverse path of the subscription. This process is repeated until a subscriber is reached.

6. Fail-Awareness in Publish/Subscribe

In order to achieve fail-awareness in a content-based publish/subscribe system, it is necessary to be able to detect and signal when it is not possible to depend upon real-time properties of lower level services due to unmasked failures. A fail-aware publish/subscribe system uses the upper bound on the transmission delay to detect any failures with respect to the timeliness of message deliveries. We have shown in the Section 4 that such detection is possible when two processes communicate with each other. However, when recalling the architecture of a content-based pub/sub system presented in Section 5, it becomes clear that two process approach is no more applicable.

There are several factors that make the computation of an upper bound on message transmission delays challenging in a content-based publish/subscribe system. First of all, timeliness properties need to be preserved between the publisher and the subscriber – being the sending and receiving ends for a publication message. This in turn implies that an upper bound on the

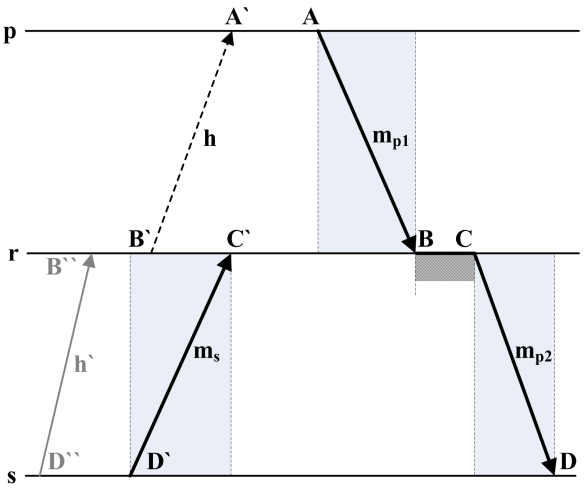


Figure 3. Calculation of an upper bound on the message transmission delay in publish/subscribe system.

transmission delay needs to be computed for messages which pass multiple nodes on their way from publisher to subscriber. Moreover, the publisher does not know which path its messages will follow.

Another issue is the fact that subscribers and publishers neither contact nor exchange any direct information. Specifically, publishers never receive any subscriptions from the subscribers. Hence, there does not exist any two way communication between them. We solve the problem of multiple nodes traversal by applying the upper bound computation in an iterative fashion (see Figure 3). Upon reception of the message m_{p1} the router r computes the upper bound $ub(m_{p1})$ on its transmission delay. Subsequently, it calculates the set of matching subscribers and/or routers m_{p1} needs to be forwarded to. Subsequently it forwards the message m_{p1} as m_{p2} to subscriber s. During the routing process r attaches following values to m_{p2}: (1) calculated upper bound $ub(m_{p1})$ and (2) upper bound on the processing time $ub_{pT}(r) = (C-B)(1+\rho_r)$ of m_{p1}. Subscriber s upon reception of m_{p2} with attached upper bound and processing time can calculate his own upper bound for m_{p2} and sum it up with attached $ub(m_{p1})$. In this way, s obtains the total upper bound on transmission delay of the publication sent by p.

The above algorithm can be applied with an arbitrary number of intermediate routers, whereas each of them executes the same algorithm as router r. There remains, however, one more issue to be solved. Namely, router r in normal course of actions does not send any messages to publisher p. When there is traffic between r and p it is not possible for r to compute any upper bounds on messages received from p (see Equation 1). Specifically, r will not be able to compute $ub(m_{p1})$.

Therefore, we send an extra empty helper message h from router r to publisher p. This allows for a successful computation of $ub(m_{p1})$. Please note that depending on the pub/sub architecture this empty helper message might not be necessary – e.g., when publishers are co-located with routers on the same nodes.

The total upper bound ub_{total} on the transmission delay of a publication sent from the publisher p and arriving at the subscriber s (messages m_{p1} and m_{p2}) can be hence expressed as:

$$\begin{aligned} ub_{imp}(m_{p1}) &= (B-B')(1+\rho_r)-(A-A')(1-\rho_p) \\ &\quad -lb_{imp}(h) \\ ub_{imp}(m_{p2}) &= (D-D')(1+\rho_s)-(C-C')(1-\rho_r) \\ &\quad -lb_{imp}(m_s) \\ ub_{pT}(r) &= (C-B)(1+\rho_r) \\ ub_{total} &= ub_{imp}(m_{p1})+ub_{imp}(m_{p2}) \\ &\quad +ub_{pT}(r) \end{aligned} \tag{7}$$

Please note that instead of using subscription m_s, one can use a potentially faster previous message h‘ (see Figure 3). This can be decided using the Equation 6.

The presented algorithm allows to differentiate between the upper bound on transmission delay and upper bound on processing time, which in turn allows to localize and pinpoint the bottlenecks in the distributed wide area system. Moreover, it does not require constant bi-directional communication. Specifically, it is enough that publisher receives only one message from the router it is connected to. The quality of the upper bound will deteriorate with time (due to the drift rate of the publishers local clock) however the computation of the upper bound will be still correct.

7. Why not just use NTP?

F-A P/S requires local clocks with a bounded drift rate (with respect to reference (real) time). We have implemented these clocks using the time stamp counter (TSC). The value of the TSC can be retrieved using the `rdtsc` assembler instruction. The return value is (on most x86 cases) a 64bit integer indicating the number of cycles which have elapsed since the last processor reset. For modern processors (clock frequencies $\approx 3GHz$) this allows for over 190 years of operation without overflow. The authors of [14] have indicated that TSCs are stable and are well suited for duration measurements. In order to verify the applicability of TSCs in the PlanetLab environment, we have conducted our own measurements (see Section 8) which allowed us to determine that the PlanetLab nodes' TSC drift rates are sufficiently stable to be used with F-A P/S.

It is important to notice that our system requires

that local clocks have only a bounded drift rate. Specifically, this means that it is possible to use any monotonically increasing counter with a bounded drift rate. This is especially important if one considers architectures which do not support the RDTSC instruction. Examples of such counters/clocks might be the 8254 Programmable Interval Timer, a High Precision Event Timer (HPET), or a ACPI Power Management Timer.

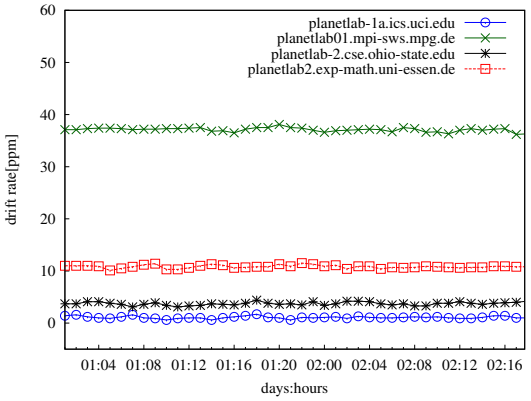
Prior to TSCs, we have been using NTP as the time reference. When reading the local clock value with the local NTP deamon it is possible to obtain the values of maximum NTP error (*maxerror*) for that reading and the local clock drift rate ($frequency$). Unfortunately, NTP does not provide any guarantees as to the values it returns. Specifically, it states that the maximum error of a clock reading is equal to the *root dispersion plus one-half the root delay* [15]. This implies that it is possible that the actual error is greater than the maximum error (*maxerror*) – if the delay for incoming and outgoing traffic to the root is asymmetric. This in turn results in the possibility of the calculation of a too small upper bound. This violates the assumption for the F-A P/S guaranteed detection of slow messages. Therefore, NTP is not well suited for the implementation of fail-aware services.

Another issue regarding NTP has been indicated in a PlanetLab mailing list [20]. It has been stated that some NTP synchronized clocks might have a drift rate as high as $130000[ppm]$[1], that is 130 milliseconds every second. This is 2600 times worse than the TSC drift rate for the worst PlanetLab host ($50[ppm]$) in our experiment.

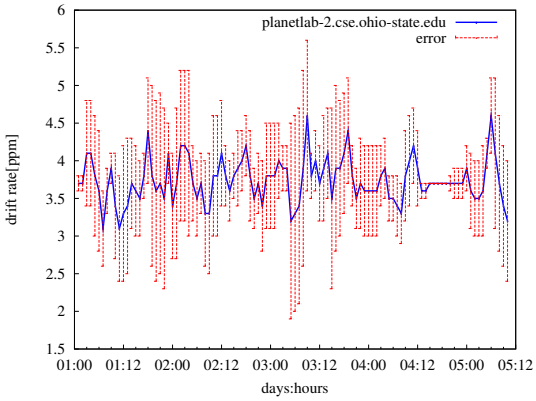
8. Results

Following the discussion in Section 7, we first present the results of our experiments concerning the drift rate of TSCs. Because the evaluation of the F-A P/S system was carried out in a PlanetLab environment, we have checked if the PlanetLab nodes used for our experiments have TSCs stable enough to allow for their use as local clocks.

In order to measure the drift rate of the TSC counter we have altered the code of the SNTP client. The modifications included using the TSC as the time source for the client application so that we were able to compare it against real time stratum 1 servers[2]. We have measured increasingly longer periods of time using both `rdtsc` and real time. Comparing both results allowed us to estimate the drift rate of the local TSC with respect to the real time. The measurements were made every hour – the minimum period that SNTP allowed for. Trying to poll the stratum 1 servers with higher rates resulted in lack of response. The results are shown in Figures 4(a) and 4(b). The relatively long measurement interval allowed us to minimize the influence of the transmission delay between the hosts and the NTP servers. We did not use SNTP servers as they are distant from the root and hence, are inaccurate [16]. It is important to mention that we have not observed any PlanetLab hosts with unstable TSCs. Since the drift rates are stable, we have shown that TSCs can be used as local clocks in the F-A P/S system. They exhibit a low, bounded drift rate and are widely available.



(a) Drift rate measurement of the TSCs for selected PlanetLab hosts



(b) The drift rate for a single host with upper and lower bound error

Figure 4. Drift rate measurements

One has to consider that PlanetLab is a specific

[1] ppm stands for one part in 10^6. A drift rate of 130000 ppm indicates that every 1 second a given clock departs 0.13 [msec] from the reference time.

[2] The NTP stratum levels define the distance from the reference clock and the associated accuracy. Stratum 1 servers are computers attached directly (e.g. via RS-232) to stratum 0 devices, such as atomic clocks or GPS clocks.

environment with mostly homogeneous hardware and software operating in a stable environment – server rooms with air conditioning. Moreover, taking under consideration the fact that out observations lasted only a month one should be careful about extending those properties to generic systems.

However, some architectures using ACPI compliant power management, e.g., AMD K8 multi-core, might expose drift issues related to performance state (P-state) and power state (C-state) changes. Suggested solution to this issue might be switching from TSC to other counters, unaffected by the power management, such as HPET or PMTimer [19]. Another problem affecting multi-core systems is context switching when subsequent `rdtsc` calls are executed on two different cores with unsynchronized TSCs. A solution to that is the new `rdtscp` instruction [2] or TSC synchronization present in current operating systems.

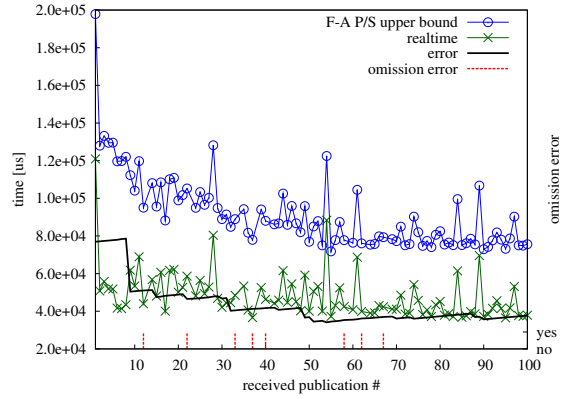
In our experiments we have not seen any influence of the power management on the measured TSC drift rates. An explanation for that fact might be that the PlanetLab nodes are mostly homogeneous systems with unified software. Since TSC is available across all PlanetLab nodes (and on most typical systems, unlike HPET) and is more precise than PMTimer we have decided to use it as our prime time source.

8.1. Results – National

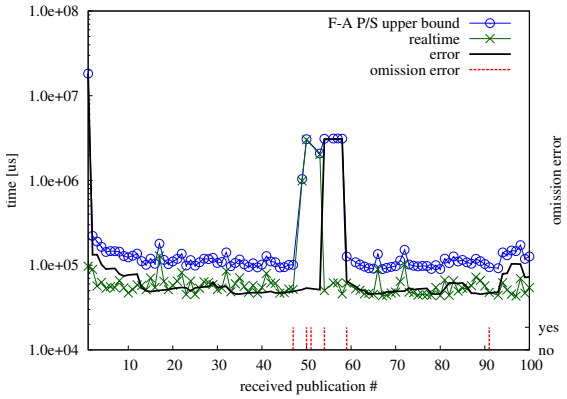
The PlanetLab environment is typically much more challenging with respect to the experienced transmission delays and omission failures than when running natively on the Internet because of the sharing of this environment by a set of users. This is ideal for our purpose because our system is targeted towards such "harsh" environments.

All communication between the nodes has been carried out using the UDP protocol. For our experiments, we have used three sets of hosts. First, two sets included *national1* and *national2* hosts. The second set included *global* hosts. *National1* and *national2* hosts were selected from within German PlanetLab nodes placed in: Berlin, Göttingen, Essen and Ilmenau (*national1*) and Bremen, Dresden, Berlin and Passau (*national2*). *Global* hosts were placed in: Delhi (India), Helsinki (Finland), Krakow (Poland) and Saarbrücken (Germany).

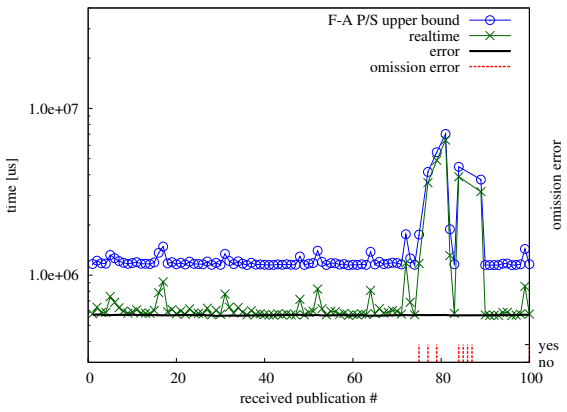
On each host we have created a single *S*IENA router process. Additionally, on each underlined host we have created a single subscriber and a single publisher process. Each underlined host acted as a starting and ending point for any publication which was also routed via remaining hosts forming a closed ring. In



(a) optimized – national1 hosts



(b) unoptimized – national1 hosts

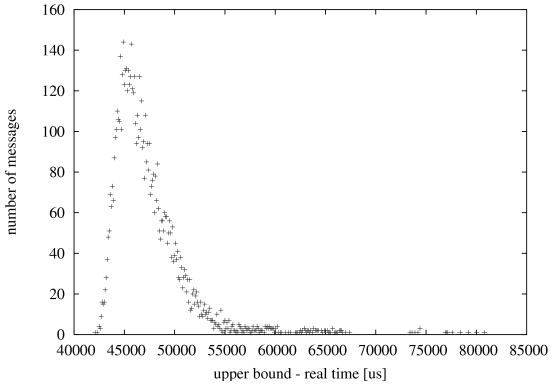


(c) optimized – global hosts

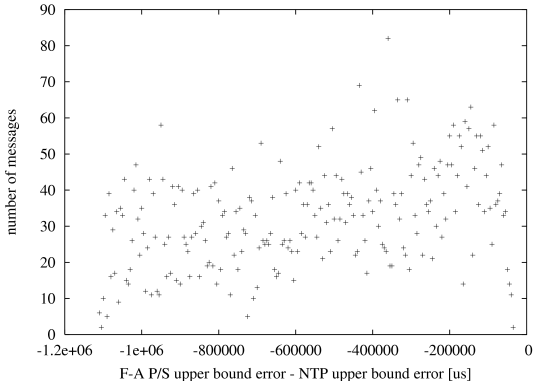
Figure 5. The upper bound on the transmission delay (including the processing time)

this way, when measuring an upper bound on the publication delay, we could compare it against the real-time transmission delay. The real-time transmission delay was calculated using the TSC of the underlined host.

Figure 5(a) shows the comparison of the calculated upper bound and the real-time transmission delay for the *national1* hosts. We can observe that F-A P/S maintains low error values (defined as difference between real time transmission delay and the upper bound) regardless of the transmission delay variations. Moreover, the helper optimization mechanisms (see Section 4) help to quickly lower the initial, high error values. For comparison, we have included the same measurement performed with an unoptimized (no helper message optimization, no lower bound on helper message computation) F-A P/S protocol version – Figure 5(b). It is clearly visible that error values depend directly on the message transmission time and are much higher than in the optimized version. Figure 6(a) shows



(a) The error of the F-A P/S upper bound (F-A P/S upper bound - real time) for the national1 hosts



(b) Comparison of the F-A P/S upper bound error and the NTP upper bound error (using maxerror) for the national1 hosts

Figure 6. F-A P/S upper bound error – national

the error of the F-A P/S upper bound calculated using TSCs. Figure 6(b) shows the comparison of the error of F-A P/S calculated upper bound and the error of the upper bound on transmission delay calculated with NTP. The delay calculated with NTP for that measurement has been bounded by using the value of *maxerror* as described in Section 7.

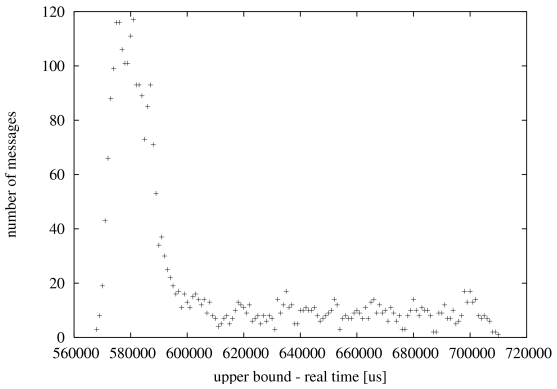
Figure 6(a) shows the distribution of the error of the calculated upper bounds. The measurement was performed for 3000 publications. The few high values are the result of the initialization phase of the protocol (see Figure 5(a)). Figure 6(b) shows that F-A P/S approach to calculate the upper bound based on the TSC delivers lower (up to 1 [s]) error values than the NTP based approach. The graph shows the difference between the F-A P/S upper bound error and the NTP upper bound error. Negative results imply lower error values of the F-A P/S upper bound.

8.2. Results – Global

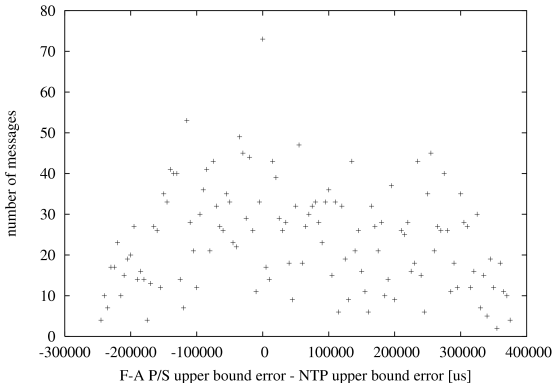
We have conducted the same set of experiments for both *national1* and *national2* hosts and *global* hosts. In this section we present selected results from the *global* hosts. First, we present the upper bound on the transmission delay measurements (Figure 5(c)). We see that despite greater upper bound and larger number of omissions the error values remain stable. Figure 7(a) shows that the error values have increased tenfold in comparison to the *national1* and *national2* runs. This is justified if we notice that the transmission times have increased twentyfold. However, looking on Figure 7(b) we notice that error on the F-A P/S upper bound is still not worse than that of the NTP upper bound. The measurement was performed for 3000 publications.

9. Summary

We have presented the Fail-Aware Publish/Subscribe (F-A P/S) – a wide area distributed system using a content-based publish/subscribe communication middleware which can guarantee detection and reporting of failures with respect to timely message delivery. F-A P/S does not require external clock synchronization nor does it impose any constraints on the used publish/subscribe middleware. Specifically, F-A P/S uses only time stamp counters (TSCs) to measure the passage of time. We show that TSCs are a reliable instrument for duration measurement, exposing bounded drift rates with respect to the real time. With F-A P/S it is possible to calculate an upper bound on the transmission delay of messages with a reasonable error in a content-based asynchronous publish/subscribe system. All our assumptions have been tested and



(a) The error of the F-A P/S upper bound (F-A P/S upper bound - real time) for the global hosts



(b) Comparison of the F-A P/S upper bound error and the NTP upper bound error (using maxerror) for the global hosts

Figure 7. F-A P/S upper bound error – global

confirmed in the PlanetLab environment.

F-A P/S performs better than when using NTP for external clock synchronization. Moreover, NTP, in contrary to F-A P/S, does not provide any guarantees as to the clock readings. Hence, it is not applicable for critical infrastructure protection applications. We believe that F-A P/S can be successfully deployed in such critical systems built from COTS components.

References

[1] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *Proceedings of the First Symposium on Network Systems Design and Implementation (NSDI)*, March 2004.

[2] R. Brunner. AMD64 status report for kernel summit. USENIX - 2004 Linux Kernel Developers Summit, July 2004.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, 2001.

[4] A. Casimiro, P. Martins, P. Verissimo, and L. Rodrigues. Measuring distributed durations with stable errors. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 310– 319, December 2001.

[5] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, October 2002.

[6] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, pages 642–657, June 1999.

[7] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9):827–850, 2001.

[8] D. Essame, J. Arlat, and D. Powell. PADRE: a Protocol for Asymmetric Duplex REdundancy. *Dependable Computing for Critical Applications*, 7:229–248, 1999.

[9] C. Fetzer. Fail-aware publish/subscribe communication in Erlang. In *Proceedings of the Fourth International Erlang User Conference*, Stockholm, Sweden, September 1998.

[10] C. Fetzer and F. Cristian. A fail-aware datagram service. In I. Bate and A. Burns, editors, *IEE Proceedings - Software Engineering*, volume 146, pages 58–74. IEE, April 1999.

[11] C. Fetzer and F. Cristian. Fail-awareness: An approach to construct fail-safe applications. *Journal of Real-Time Systems*, pages 203–238, March 2003.

[12] K. H. Gjermundrød, I. Dionysiou, D. Bakken, C. Hauser, and A. Bose. Flexible and robust status dissemination middleware for the electric power grid. Technical Report EECS-GS-003, School of Electrical Engineering and Computer Science Washington State University, Pullman, Washington 99164-2752 USA, September 2003.

[13] J. Kaiser and M. Mock. Implementing the real-time publisher/subscriber model on the controller area network (CAN). *isorc*, 00:172, 1999.

[14] H. Marouani and M. Dagenais. Comparing high resolution timestamps in computer clusters. In *Canadian Conference on Electrical and Computer Engineering*, pages 400–403, 2005.

[15] D. L. Mills. Network time protocol (version 3) specification, implementation and analysis. RFC 1305, March 1992.

[16] D. L. Mills. Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI. RFC 4330, January 2006.

[17] A. Mislove, A. Post, A. Haeberlen, and P. Druschely. Experiences in building and operating ePOST, a reliable peer-to-peer pplication. In *EuroSys*, 2006.

[18] P. Murray. A Distributed State Monitoring Service for Adaptive Application Management. *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 200–205, July 2005.

[19] B. Nagendra. AMD TSC Drift Solutions in Red Hat

Enterprise Linux. Online, December 2006.

[20] A. Nakao. Timing Problem on PlanetLab (bad NTP). PlanetLab users mailing list, March 2004.

[21] G. Pardo-Castellote. DDS Spec Outfits Publish/Subscribe Technology for the GIG. *COTS Journal*, 4, April 2005.

[22] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content Based Routing with Elvin4. *Proceedings AUUG2K, Canberra, Australia, June*, 2000.

[23] W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. Buchmann. A peer-to-peer approach to content-based publish/subscribe. *Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8, 2003.

[24] The Executive Office of the President, The Department of Homeland Security. The national plan for research and development in support of critical infrastructure protection, April 2005.

[25] K. Tomsovic, D. Bakken, V. Venkatasubramanian, and A. Bose. Designing the next generation of real-time control, communication and computations for large power systems. *Proceedings of the IEEE – Special Issue on Energy Infrastructure Systems*, 93(5):965–979, May 2005.

[26] Y. Zhao, D. Sturman, and S. Bhola. Subscription propagation in highly-available publish/subscribe middleware. *Lecture Notes in Computer Science*, 3231:274 – 293, 2004.