Handling Overload in Publish/Subscribe Systems

Zbigniew Jerzak, Christof Fetzer
Dresden University of Technology
D-01062 Dresden, Germany
Telephone: ++49 351 463–39708
Fax: ++49 351 463–39710
Email: {Zbigniew.Jerzak, Christof.Fetzer}@tu-dresden.de

***Abstract*— This paper proposes a new approach for handling overload in Publish/Subscribe systems. We focus on the fact that every service has to cope with the limitations imposed by the external environment (e.g., network congestion) and the limitations resulting directly from within the service itself – e.g., the maximum available computational power. This work seeks to aid Publish/Subscribe mechanisms with the ability to handle both aforementioned constraints in a graceful way, simultaneously ensuring for the most valuable information to be given the highest chance of a successful delivery.**

I. Introduction and Motivation

A. Background

Publish/Subscribe (P/S) systems are a communication model where messages are not given explicit destination addresses, but they are determined by matching the content of the message against selection predicates declared by the nodes [1]. Information produced at publishers is routed through the P/S system in form of publications. Subscribers (nodes willing to obtain the information) express their interest by forming and issuing subscriptions. Publication is a set of typed attributes. Each attribute is identified by a name, it has a type and a value. Subscription is a disjunction of conjunctions of constraints on individual attributes. Each constraint has a name, type an operator and a value. Whenever a subscription $s1$ expresses interest in a superset of information selected by a subscription $s2$, it is said that $s1$ covers $s2$. The coverage allows to limit the dissemination of subscriptions. Clearly, if a subscription $s1$ was forwarded on a link $l1$ we do not need to subsequently forward the subscription $s2$ which is covered by $s1$. Aggregation is a process when subscription $s1$ covers $s2$.

B. Motivation

Contemporary distributed systems are subject to frequent load peaks [2], link failures and node crashes. This in turn leads to overloads in other parts of the system, originally unaffected by the failure. The cause for that is an increased load on the available system components, as the same load has to be handled using a smaller amount of resources. Another issue is a "snowball effect" when one overloaded part of the system triggers overload in following ones [3].

This paper seeks to aid the existing research in the field of Publish/Subscribe systems and the content based networking [1], [4] with mechanisms which will allow to cope with overload conditions. We propose an extension of the content based networking systems which handles the issues of link and router/node overload. By link overload we understand the situation when the link is subject to greater load than it is capable of transmitting. Router/node overload occurs when a node responsible for the forwarding/routing the data on/between different parts of the network cannot cope with the processing of the information it receives.

The system we have designed – the Highly-Available Publish/Subscribe (HAPS) – has been carefully crafted so as not to violate the very nature of the Publish/Subscribe systems, being the *time*, *space* and *synchronisation* decoupling [5]. The decoupled nature of the Publish/Subscribe systems constitutes the foundation of its scalability and flexibility, therefore we believe that upholding these properties is the essential part of any P/S modification. Moreover, we introduce the notion of fully scalable priorities/prices allowing for a detailed control of the information flow and an intelligent information shedding. This quantitative information allows us to better cope with the overload condition in the P/S system by providing a detailed insight into the current network conditions and participants requirements.

We provide a proof-of-concept C++ implementation and confirm its applicability with tests using the OMNeT++ environment. To the best of our knowledge, no similar research has been undertaken so far.

II. Related Work

Publish/Subscribe systems have recently drawn considerable attention. They have evolved from relatively simple and static topic-based systems to complex and powerful content-based solutions [6], [7], [8]. However, most applications and derived services do not utilize the flexibility of the fully decoupled publish/subscribe design. In the case of Astrolabe [9] this is due to the need for hierarchical system architecture and gossip based status updates. The GridStat approach, described in [10], uses mixed peer-to-peer and hierarchical structures in order to enforce the service level agreements on the parameters of the information delivery, which yields scalability problems. We believe that in order to take full advantage of the publish/subscribe model, one should not impose any requirements that are in contrast with its decoupled nature. We believe that the content-based networking and the approach proposed in [1], [4] are the closest ones to the publish/subscribe nature. The aforementioned work and more recent contributions [11] do not, however, address the issues of overload and the P/S communication systems yield[1]. The idea of the HAPS system has been influenced by the self-regulating market approach presented in [13] and [14].

III. Handling Overload

When coping with link congestion or router computational limitations we must pay attention so as not to violate any of the *time*, *space* and *synchronisation* P/S decoupling properties. We believe that when facing system overload it is not possible to provide unaffected service to all the users all the time. It has been proven that every datagram network with infinite storage, FIFO queueing and a finite packet lifetime will under overload drop all packets [15]. Hence, it is obvious that attempts to deliver service to all the requesting parties in case of

[1]Following [12] we define yield as the fraction of the queries that are completed. $\text{yield} = \frac{\text{queries completed}}{\text{queries offered}}$

an overload will lead to an ungraceful degradation of the service. On the other hand, due to the very nature of the P/S, it is not possible to perform a priori link reservation as proposed in [10] and eventual load shedding without violating the synchronisation decoupling property. Therefore we opt for a priority based method of handling overload which seamlessly integrates with the underlying P/S system without violation of the decoupling properties. Moreover we can make our congestion handling decisions in the same manner as the routing is performed, namely *content based*. Based on the content's importance indicated by the client we are able to determine which messages represent the high value for the subscribers. Moreover we can use this information to efficiently aid our decision process as far as network management is concerned.

A. Price/Priority System

In our HAPS system, subscribers compete with each other by providing prices describing their willingness to pay for information. The price which is propagated along with the request for the information indicates the maximum amount a client is willing to pay for the reception of that information. When there is no congestion and there are no computational limitations in the network, subscribers are charged a flat rate for their access. However, as soon as there is a need to perform load shedding or to handle network congestion, the routers which forward the information, will choose to skip the links (*router overload*) or the messages (*link overload*) which will be the least "profitable" from the network's point of view, i.e., would result in the smaller revenue/profit. In the case of multiple routers propagating the information between the publisher and the subscriber, the prices are aggregated on the links in order to allow for recognition of the most "profitable" ones.

It is important to note that the notion of the maximum price introduced above can be simultaneously seen as an open priority system. Traditional prioritizing systems make use of a fixed number of available priority classes, partitioning the whole content space so as to assign each message/node/process to one of the available classes. Our approach differs in that we are not limited by a fixed number of available classes. Subscribers can always assign a priority higher or lower than any of the ones currently present in the system, being limited only by the data type used for representing the priorities. This, in turn, gives us a greater degree of freedom in case of the overload when the priorities might change dramatically.

HAPS makes use of two load shedding strategies. The first one focuses on handling of the router overload (when the load exceeds router's computational capacity) and involves temporary removal of entries from the router's routing table. The second strategy, described in Section III-C, focuses on shedding messages on the overloaded links. Introducing those two strategies we cover two common types of failures in a distributed system: node overload and link saturation failures. The link shedding strategy focuses on alleviating the load imposed on the overloaded routers. Link shedding allows us to decrease load in the router and hence prevent ungraceful service degradation. The message shedding protects the network links from overload and random message drops and delays. We have designed algorithms which allow routers to decide which messages should be forwarded and which should be dropped. The resulting policies ensure that the messages with the highest global value have simultaneously the greatest chance of being forwarded in the case of link overload.

B. Handling Router Overload

Each router stores a table with all the subscriptions it has received – see Figure 1. The table associates subscriptions and the links they

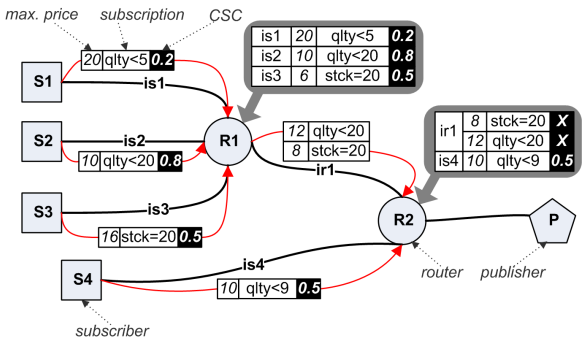

Fig. 1. Routing of subscriptions. Router R1 aggregates the $qlty < 5$ and $qlty < 20$ subscriptions.

have arrived from, using the link id as an index for the subscriptions. When removing an entry denoted by the specified link id we disclose all the subscriptions that have arrived on that link from the matching process. This in turn reduces the time needed by the router to search its tables and perform message matching.

HAPS routers adapt to the changing load in the system in that the dropped links can be reintegrated when the load on the router drops below a threshold defined as a function of the maximum processing delay. In order to avoid a high amount of oscillation (shedding and reintegration of links) HAPS routers use *shedding timeout* – a value determining how often can a link be shedded or reintegrated.

In order to be able to calculate the load imposed on the router we introduce the computational overhead value: $\Delta(m)$. We have to calculate the value of the $\Delta(m)$ "by hand" due to the fact that we test our system in a discrete simulation environment, which does not introduce the notion of computational delay. If the router would have been running on a physical computer, one would use the current CPU load value as a measure for computational overhead. We calculate the $\Delta(m)$ (see Eq. 2) for the current message m as a function of the number and complexity of the subscriptions stored in the router's routing tables.

Based on the results presented in [4] we have assigned a fixed delay ($\Delta_1 = 5{\cdot}10^{-8}\,[s]$) to a comparison of a single constraint subscription and a single attribute publication[2]. Knowing the total numbers of comparisons (N) that were necessary to forward the message we can trivially calculate the current message processing delay (Δ_{curr}, see Eq. 1). The so calculated Δ_{curr} is then averaged with a predefined weight – w. The averaging process is necessary as the matching times often vary by more than an order of magnitude due to the coverage of subscriptions. The shedding timeout (sT, see Eq. 3) is calculated by the router as a product of an average message arrival time (δ) and the inverse of the weight value used in the weighted average function:

$$\Delta_{curr} = \Delta_1 \cdot N \quad (1)$$

$$\Delta(m) = (1-w)\cdot\Delta(m-1) + w\cdot\Delta_{curr} \quad (2)$$

$$sT = \delta\cdot\frac{1}{w} \quad (3)$$

$\Delta(m-1)$ denotes the router computational overhead for the previous message.

C. Handling Link Overload

The second load shedding strategy is invoked whenever the router encounters congestion on any of its outgoing links. For this strategy

[2]This applies also to comparison of two single constraint subscriptions.

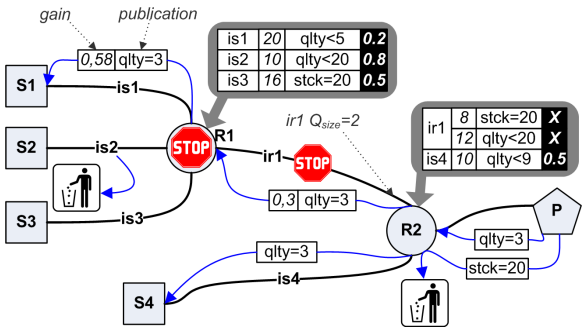

Fig. 2. Forwarding of publications. Router R2 knows that link ir1 is overloaded and drops the $stck = 20$ message in favour of the $qlty = 3$. The overloaded router R1 eliminates the link is2 from its routing table and does not forward the $qlty = 3$ message to s2.

we assume that routers are capable of estimating the available bandwidth on any of their outgoing links.

HAPS routers store the outgoing messages in a prioritized queues assigned to corresponding outgoing links. The queue output rate is limited using a token bucket algorithm with token rate initially equal to the link bandwidth. All links and queues attached to them are monitored by the router which has been programmed with a modified RED [16] algorithm. When the size of the queue (and thus the transmission delay for message stored in it) increases beyond a given *drop_start* threshold, we start dropping messages with an increasing probability. The drop probability approaches one as the expected transmission delay approaches the predefined maximum delay threshold *drop_all*. When calculating the drop probability for a given message HAPS routers account for the message priority – see section IV-B. Seen from a high level perspective, message priority is a measure of interest the P/S participants have in a given message and the value a given message represents for them.

IV. The Protocol

For brevity of the protocol description presented herewith we consider only single attribute messages and single constraint subscriptions. We do not impose this limitation in the implementation of the protocol in the OMNeT++ environment.

A. Routing and Forwarding

Publications are broadcasted to the network with broadcast distribution tree branches pruned by the subscriptions constraints. HAPS extends the standard subscriptions with the notion of their importance, or, more correctly, with the notion of importance of data that these subscriptions should trigger. Therefore, each subscription piggybacks a maximum price and a Content Space Coverage (CSC) – see Figure 1. The maximum price represents an amount a subscriber is willing to pay for each received publication matched by this subscription. We will describe the CSC in Section IV-A.1 – Content Space Coverage. Introducing the CSC in depth now, would result in forward references to the following material.

HAPS routers use the coverage mechanism to limit the dissemination of subscriptions. Therefore, we need to take into account its influence on the maximum price value of a subscription s that is forwarded by the router and covers other subscriptions already present in the router's routing table. Hence whenever a subscription s covers other subscriptions, its maximum price has to be recalculated before forwarding as:

$$\sum_{i:i\in A} c(i)p(i) \tag{4}$$

where A is a set of all subscriptions covered by s, $c(i)$ is the coverage declared in the ith covered subscription and $p(i)$ is the corresponding maximum price.

In order for the actual "charging" (charging implemented in routers is a quantitative measure of the shedding policies performance) to take place, publications in HAPS piggyback a gain value – G (see Fig. 2). The gain value, initially set to zero, describes how much of the original advertised maximum price should the subscriber be charged for the delivered publication.

Iff there is a need to drop messages or perform load shedding in the router it computes the local gain values: G_{l1} (see Eq. 5) or G_{l2} (see Eq. 6). Seen from a high level perspective gain value indicates how much preference was the publication carrying it given over the other publications, and thus how much more valuable has it become. The intuition behind that mechanism is that the information which has been provided at the cost of other information should also be more expensive:

$$G_{l1} = \frac{\sum_{i:i\in E} p(i)}{\sum_{j:j\in N} p(j)} \tag{5}$$

$$G_{l2} = \frac{\sum_{i:i\in F} p(i)}{\sum_{j:j\in N} p(j)} \cdot \frac{1}{Q_{size}} \tag{6}$$

In case of router overload, it computes G_{l1}, where E represents the set of all subscriptions on the links excluded from consideration (shedded), and N represents the set of all subscriptions on all outgoing links. The $p(i)$ represents the maximum price for subscription i. G_{l2} is computed in case of link overload. F represents all the subscriptions on the given link in the routing table which are matched by the publication that is forwarded. N represents the set of all subscriptions on the given link. Q_{size} represents the size of the router's buffer – we assume that the messages have the similar size. The link and message shedding is represented by routers R1 and R2, correspondingly. See Figure 2.

The second step in calculation of the gain value (G, see Eq. 7) is the inclusion of the current gain value (G_{old}) piggybacked by the publication:

$$G = G_{old} + (1 - G_{old}) \cdot G_l \tag{7}$$

The G_l represents either G_{l1} or G_{l2}. If the router had to perform both link and message shedding it first performs message shedding and updates the gain value according to the Equations 5 and 7. Subsequently the message shedding and gain value updates are performed, according to Equations 6 and 7.

Every HAPS router stores charges in a so called *billing table* which correlates the outgoing links with the charges that have been calculated for the publications sent on them. The gain values on publications increase with their flow through the routers which perform load shedding, hence the final price a subscriber will be charged for receiving a publication is calculated by the last router delivering the message directly to the subscriber[3]. It is a product of the gain value (G) piggybacked on the publication and the maximum price ($maxprice$) declared by the client and stored in the client's subscription:

$$\text{charged amount} = G \cdot maxprice \tag{8}$$

[3]Please note that this and all other algorithms described in this section are fully symmetric.

1) Content Space Coverage: The Content Space Coverage (CSC) helps the HAPS system to cope with the problem of determining how many users will actually receive a publication if it is propagated on the given link. If we consider the scenario similar to that of Figure 2 where the publisher P publishes a message $qlty = 8$, we can see that the subscriber $S1$ will not receive it, although its maximum price was taken under consideration when computing the new maximum price for subscription $qlty < 20$, see Fig. 1. Therefore, if the subscriber $S1$ would specify a very high maximum price on his narrow subscription, router $R2$ would forward publications on link $ir1$ hoping to receive payoff from delivery to $S1$. However, only a few of those publications would be actually forwarded to $S1$, just like in case of $qlty = 8$ publication.

The problem we have to tackle arises from the fact that subscription coverage "hides" the detailed information about the subscribers (and their detailed interest along with declared maximum prices) downstream from the router receiving the aggregated subscription. In our aforementioned example router $R2$ does not know that forwarding a publication p matching the subscription $qlty < 20$ it has only 25% chance that p will be forwarded to $S1$. Therefore we introduce Content Space Coverage value allowing us to correlate the declared maximum price and the probability that the publication matching the widest aggregated subscription will be forwarded to a given subscriber. The responsibility of providing this value lies on the subscriber. The subscriber can query the router for the widest subscription covering his subscription and thus is capable of calculating an estimate of the CSC himself.

In order to prevent selfish subscribers from specifying arbitrarily high CSC values, HAPS charges a subscriber for *every* publication p arriving at the router to which it is connected. This charging occurs only for the subscribers whose subscription does not match p but is covered by another subscription matching p – see the following example. Therefore we modify the Equation 8 presented in the previous section.

Considering the aforementioned example with $qlty = 8$ publication: the subscriber $S3$ will not be charged at all, the subscriber $S2$ will be charged the following amount:

$$\text{charged amount} = G \cdot CSC \cdot maxprice \tag{9}$$

and the subscriber $S1$ will be charged the following amount:

$$\text{charged amount} = G \cdot maxprice \tag{10}$$

G and $maxprice$ have been introduced in Equation 8. The CSC is the Content Space Coverage declared in the subscription.

B. Load Shedding Mechanisms

Two aforementioned shedding mechanisms require routers to be capable to asses the profitability of both links and messages they manage/forward. The sole purpose of the introduced charges is to allow the router to decide which message has the highest or lowest priority and hence decide about it shedding or forwarding. This is a non-trivial task, especially if we keep in mind the fact that the P/S communication scheme is fully decoupled and there can be potentially hundreds of subscribers interested in the given message that are unknown to the router and vice versa. For detailed performance comparison of different load shedding mechanisms please refer to Section V.

For link shedding, HAPS router first calculates the average weighted delay (Eq. 2) and checks whether it exceeds the drop threshold. It also checks whether the shedding timeout (Eq. 3) allows for the link to be dropped. If both conditions evaluate to true a HAPS router must select a link to be shedded. The selection of the link proceeds as follows. For each outgoing interface in the routing table a HAPS router calculates the sum of the advertised maximum prices on the stored subscriptions. If a subscription has been sent directly from subscriber (no intermediate routers) the maximum price on the given subscription is multiplied by the CSC value declared in the same subscription. HAPS router chooses to shed a link with the lowest sum. Seen from a high perspective, we calculate a measure indicating which link connects the subscribers willing to pay the most for the publications.

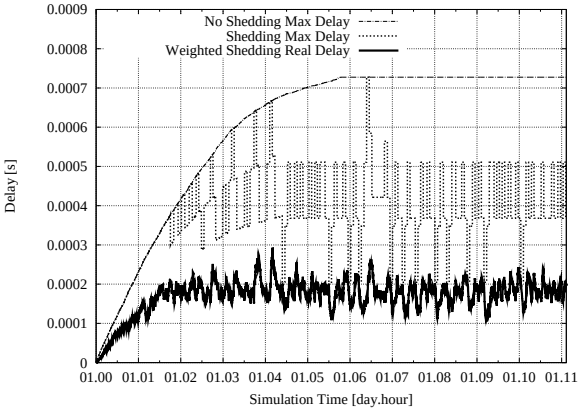

Fig. 3. Adaptive link shedding. Shedding Timeout = $2 \cdot 10^{-4}$ $[s]$

Message shedding is a fine grained mechanism requiring additional algorithms and data structures built into the HAPS router. If we consider the example from Figure 2, we can clearly see that router R2 has to be able to see both messages $quality = 3$ and $stock = 20$ prior to deciding about forwarding one of them in favor of the other. We achieve this by implementing a queue for every outgoing link. Currently the length of the queue is limited only by the link throughput. It has been shown that this approach allows for reasonable[4] queue lengths [15]. The first decision to be made by the router is whether to shed a publication or a subscription. Currently we use a fair algorithm which calculates the probability of drop of either type of message as a liner function of their respective count in the queue. In order to shed messages intelligently, the queue control algorithm searches for the messages with the lowest potential value. For publications this is identified by the sum of prices on the matching subscriptions. For subscriptions we use the declared maximum price. Please note that the value G_{l1} (see Eq. 5) can be calculated for a given message only once. In contrast the G_{l2} (see Eq. 6) is calculated every time a message is dropped from a queue up to Q_{size} times. This type of ongoing calculation allows us to take into account arrivals of new subscriptions which are stored in the routing table and influence the value of the messages already present in the queue.

V. Experiments

We have created a proof of concept implementation of the HAPS protocol in C++ and tested its behavior in the OMNeT++ simulation environment. Our simulation environment consisted of twenty nodes and in each simulation we have generated approximately $2 \cdot 10^6$ messages. For all the tests we have instrumented publishers/subscribers to send a varying number of publications/subscriptions, according to

[4]The buffer space required for 10Mb Ethernet with an upper bound on the time-to-live of 255 is 318 million bytes – see RFC 970 for details.

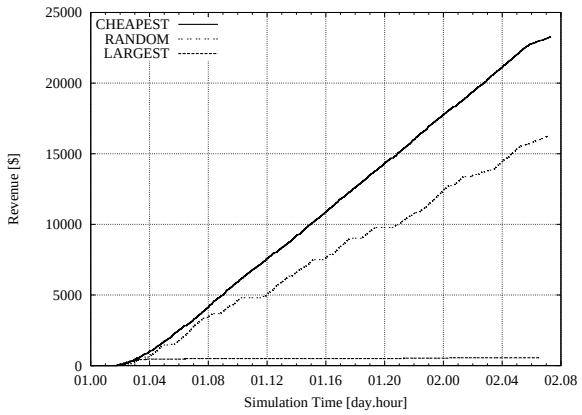

Fig. 4. Router revenue for different link shedding techniques.

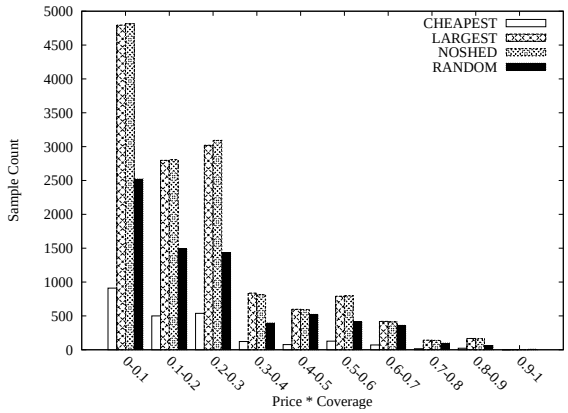

Fig. 5. Distribution of received publication prices for subscriber *node04*.

normal distribution with standard deviation value equal to 25% of the mean value. The number of publications sent is greater than the number of subscriptions (approximatelly 100%), as we also want to inspect the system behavior under "stable" conditions.

First part of our tests has focused on the applicability of link shedding for handling overload. Figure 3 illustrates the shedding process in the router. The Y axis represents the computational overhead in the router. *No Shedding Max Delay* indicates the delay for processing of a single message when no link shedding is used and message is matched against all subscriptions in the routing table. *Shedding Max Delay* is similar to the previous except that the value has been calculated with shedding enabled. The *Weighted Shedding Real Delay* is the averaged actual matching time of a single message, see Eq. 2. It can be seen that the actual delay in the router oscillates around the predefined shedding Timeout (see Eq. 3) equal to $2 \cdot 10^{-4}\,[s]$. The *No Shedding Max Delay* value ceases to grow at Simulation Time $\approx$ 01d 6hrs due to the fact that we generate greater number of publications. Without new subscriptions maximum matching time remains constant.

Figure 4 illustrates the revenue calculated on the link connecting a subscriber to the router for different link shedding techniques. The *CHEAPEST* strategy calculated the potential link value by summing up all prices on subscriptions stored in the routing table under this link. The *LARGEST* strategy was used to shed links with the smallest number of subscriptions stored for them. The *RANDOM* strategy dropped a random link. It can be seen that the *CHEAPEST* strategy implemented using HAPS extension outperforms other ones.

Figures 5 and 6 represent the distribution of the prices on subscriptions that triggered the reception of the publications. The Price value spans [0.1, 10) and the coverage spans [0.05, 0.1). Hence the possible distribution of Price * Coverage spans [0.005, 1). It can be seen that for the *CHEAPEST* strategy nodes with higher amount of low paid subscriptions (Fig. 5) are receiving lower number of publications than the nodes with higher amount of well paid subscriptions (Fig. 6). This intuitively confirms the *CHEAPEST* strategy used at the router. *NOSHED* is a reference value measured when no link shedding has been used.

Figure 7 shows how different strategies used for message shedding influence the revenue at the router. In all cases we always shed a message which has the lowest value in the queue according to the algorithm used for shedding. If we choose to shed a subscription (see Section IV-B), we use an algorithm calculating the subscription value as a sum of its price and the prices of subscriptions covered

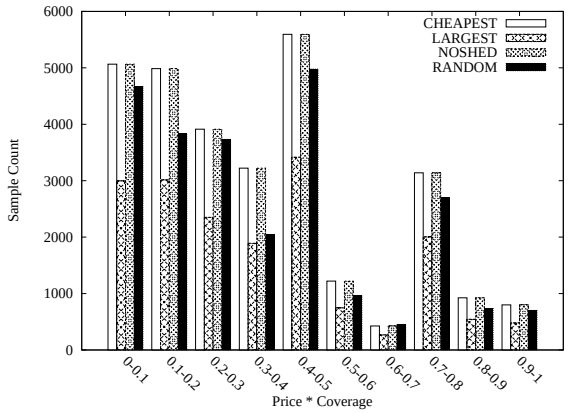

Fig. 6. Distribution of received publication prices for subscriber *node13*.

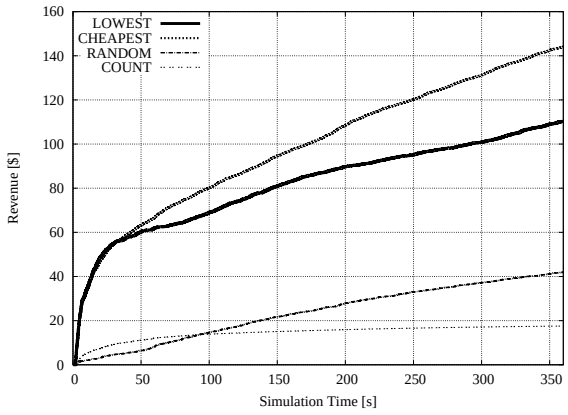

Fig. 7. Router revenue for different message shedding techniques.

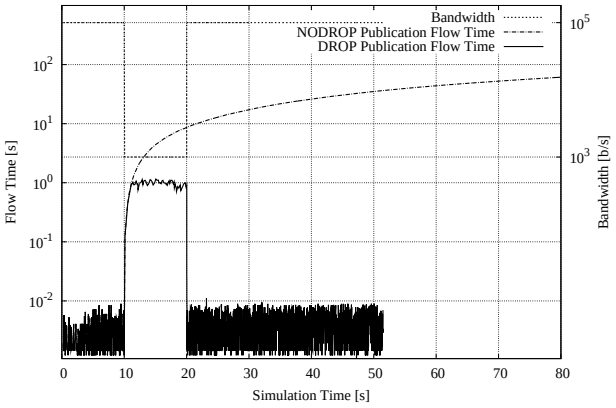

Fig. 8. Link failure and resulting publication flow times.

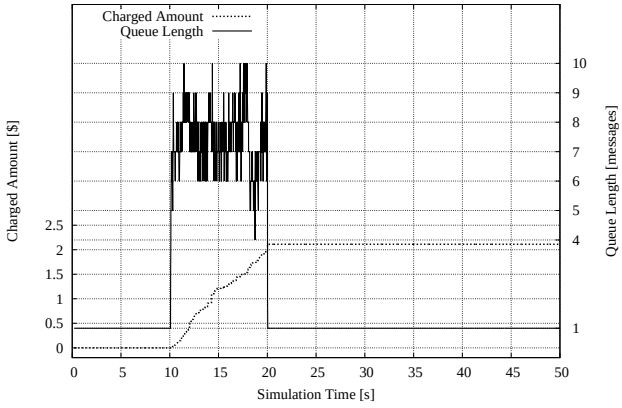

Fig. 9. Queue length and revenue at the router.

by it. For publications we compare the following four strategies: *CHEAPEST* – uses the sum of prices on the matching subscriptions, *LOWEST* – uses the gain piggybacked on the publication, *COUNT* – uses the number of matching subscriptions and *RANDOM* – selects a random subscription. We see that *CHEAPEST* strategy generates the most revenue, however, it has to be noted that it represents a certain computational overhead. The *LOWEST* strategy is cheaper in terms of CPU cycles, however because the gain value is influenced by the values of matching subscriptions the revenue loss is not as large as with the two other strategies. We plan on using both *CHEAPEST* and *LOWEST* strategies for the HAPS router, depending on the available computational power in the router.

Figure 8 simulates a link failure when the available high bandwidth link (100 $[\frac{kbit}{s}]$) goes out of service and is being temporarily replaced by a low bandwidth backup (1 $[\frac{kbit}{s}]$). We can see that with no message shedding (NODROP Publication Flow Time) the effect on the message latency is devastating and has a long term effect on the network. With HAPS (DROP Publication Flow Time) the latency does not exceed the predefined threshold value ($1[s]$) and returns to normal as soon as the primary link has resumed service. Figure 9 indicates that the router calculates the revenue only for the overload situation and that the queue length for the outgoing overloaded link returns to normal as soon as overload period is over.

VI. Conclusions and Future Work

It has been shown that the current HAPS version is a solid foundation for further evaluation of the robust P/S system. We have provided a solution to the problem of smart overload handling whilst maintaining the decoupled properties of the P/S systems. Moreover, we have shown how to cope with two prevalent types of common failures in event driven distributed systems, i.e., the node and link failure. We have also provided experimental results which confirm the applicability of our approach. We are currently working on refining our routing strategies with new algorithms and we focus on extending the prices/charges model so as to create a solid foundation for a new market-driven P/S system.

References

[1] A. Carzaniga, M. J. Rutherford, and A. L. Wolf, "A routing scheme for content-based networking," in *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, March 2004.

[2] D. Ibison, "Tokyo exchange overwhelmed by panic trading," *Financial Times*, 18th January 2006.

[3] F. L. Alvarado and R. Rajaraman, "The 2003 blackout: Did the system operator have enough power?" August 2003. [Online]. Available: http://www.ksg.harvard.edu/hepg/Standard_Mkt_dsgn/Blackout_Investigation_lrca.pdf

[4] A. Carzaniga and A. L. Wolf, "Forwarding in a content-based network," in *Proceedings of ACM SIGCOMM 2003*, Karlsruhe, Germany, Aug. 2003, pp. 163–174.

[5] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[6] Y. Zhao, D. Sturman, and S. Bhola, "Subscription propagation in highly-available publish/subscribe middleware," in *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., October 2004, pp. 274–293.

[7] R. Chand and P. Felber, "Semantic peer-to-peer overlays for publish/subscribe networks," in *Proceedings of the International Conference on Parallel and Distributed Computing (Euro-Par'05)*, Lisboa, Portugal, August 2005.

[8] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Trans. Comput. Syst.*, vol. 19, no. 3, pp. 332–383, 2001.

[9] R. V. Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 164–206, 2003.

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

[11] R. Zhang and Y. C. Hu, "Hyper: A hybrid approach to efficient content-based publish/subscribe," in *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 427–436.

[12] E. A. Brewer, "Lessons from giant-scale services," *IEEE Internet Computing*, vol. 5, no. 4, pp. 46–55, 2001.

[13] J. K. MacKie-Mason and H. R. Varian, "Pricing the internet," prepared for the conference Public Access to the Internet, JFK School of Government, May 1993.

[14] A. Blanc, Y.-K. Liu, and A. Vahdat, "Designing incentives for peer-to-peer routing," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 1, March 2005, pp. 374 – 385.

[15] J. Nagle, "On packet switches with infinite storage," *Communications, IEEE Transactions on [legacy, pre - 1988]*, vol. 35, no. 4, pp. 435–438, 1987. [Online]. Available: http://ieeexplore.ieee.org/iel5/8159/24018/01096782.pdf?tp=&arnumber=1096782&isnumber=24018

[16] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, 1993.