Full Citation:

Alberto Cortes, Carlos Garcia-Rubio, Celeste Campo, Andres Marin, Florina Almenarez, Daniel Diaz. "Decoupling path failure detection from congestion control to improve SCTP failovers", IEEE Communication Letters, volume 12, issue 11, November 2008, pages 858-860, ISSN: 1089-7798, DOI: 10.1109/LCOMM.2008.080884

Decoupling Path Failure Detection from Congestion Control to Improve SCTP Failovers

Alberto Cortés, Carlos García-Rubio, Celeste Campo, Andrés Marín, Florina Almenárez, and Daniel Díaz

***Abstract*—SCTP handover is too slow to be useful as a mobility solution for most applications. With quicker handovers, SCTP will be an interesting solution to mobility. SCTP uses retransmissions as probes for path failure detection. Congestion control forces slow paced retransmissions but quick failovers need fast paced probes. Our solution is to stop using retransmissions as probes. Instead, we propose active path monitoring using unreliable heartbeats. Applications can tune this algorithm to meet their own requirements. We show data from a simple experiment in a real environment.**

***Index Terms*—Computer network reliability, congestion control, transport protocols.**

I. Introduction

SCTP is a reliable, connection oriented, transport protocol. It shares some characteristics with TCP and UDP but also provides some new functionality. SCTP is described in [1].

SCTP supports multihoming hosts: the endpoints of an SCTP connection can have multiple IP addresses that allow for redundant network paths. Each host has a primary address; during normal operation, an SCTP endpoint sends the data and control messages to its correspondent primary address.

In case of a path failure, the endpoints will revert to use an alternative address, hoping to circumvent the offending path. The whole failover process does not require application intervention. This automatic failover provides path failure reliability and mobility on overlapping scenarios.

SCTP takes between 63 seconds and 6 minutes to make a failover; this is too slow for most applications. The dominant term of this failover time is the time needed to detect a path failure as switching paths is instantaneous.

In this letter we describe:

- Why it's difficult to tune path failure detection (PFD for now on) to adapt the failover process to different application requirements.
- How decoupling PFD from congestion control can solve those tuning problems, allowing for new and interesting PFD algorithms.
- How a very simple and conservative PFD algorithm based on active path monitoring using non-reliable probes can safely adapt the failover process to your application requirements.

We also present experimental data figures from a simple real environment.

Manuscript received June 8, 2008. The associate editor coordinating the review of this letter and approving it for publication was J. Murphy.

The authors are with the Department of Telematic Engineering, University Carlos III of Madrid, Spain (e-mail: {alcortes, cgr, celeste, amarin, florina, dds}@it.uc3m.es).

Digital Object Identifier 10.1109/LCOMM.2008.080884

II. How path failure detection works

The PFD algorithm is very simple: 5 consecutive retransmissions trigger a path failure detection. This maximum number of consecutive retransmissions is called *Path.Max.Retrans* or PMR and it's a configurable parameter on many SCTP implementations.

Retransmissions provide reliability in case of lost or damaged data. A retransmission takes place when SCTP has been waiting *too long* for an acknowledgment to a message. *Too long* means more than it should take a message to go forth and its acknowledgment to come back. This time is called the *round trip time* or RTT of the path and can change unexpectedly during the life of a connection.

SCTP estimates RTT based on measurements taken on previous acknowledgments in a similar way to TCP[1]. To cope with retransmission ambiguity[2], acknowledgments from retransmitted messages aren't used for this estimation; this is known as the Karn/Partridge algorithm.

In the case of a sudden increase in the path's RTT, an artificial increase of the estimation is needed to stop retransmissions and begin fetching new data to the estimation algorithm. The Karn/Partridge algorithm doubles the last estimated RTT each time there is a retransmission. Eventually, this exponential backoff should deal with the underestimation. This means that successive retransmissions are exponentially delayed in time.

This exponential backoff in combination with the PMR mechanism and SCTP default parameters give a path failure detection time between 63 seconds and 6 minutes. See [5] for an excellent analysis of this topic.

Counting retransmissions is a simple and effective way to distinguish a real path failure from an unexpected delay if you are a protocol designer. But for the application developer, this algorithm presents some problems. How long will my application stall trying to decide if there has been a path failure? Is this delay acceptable for my application? Is there anything the application can do to control this delay? Is there any other figures SCTP can use to trigger a failover? Goodput, for example?

Even SCTP best PFD time, 63 seconds, is too much for many internet applications. There is nothing the application can do to control that delay and there is no other way to trigger a failover.

[1] see [2] and [3] for details

[2] see [4]

III. Decoupling path failure detection from congestion control

Our proposal is to stop using retransmission counting as the (only) source of information for PFD. By decoupling PFD from congestion control, there will be room for new PFD algorithms, targeted to each application requirements: from aggressive active probing to passive monitoring of transmission parameters like delay or goodput.

We have setup an experiment to prove that such decoupling is feasible. We have used a very simple PFD algorithm based on active monitoring. We won't present or discuss advanced PFD algorithms in this letter, as it will be the topic of future research.

In a sense, SCTP is already using active monitoring for PFD; retransmissions are used as monitoring probes. As PFD is not really interested in the (often long and exponentially delayed) retransmissions, but just in detecting a path failure, we claim that probing with lighter and more frequent probes is more indicated.

A heartbeat is an SCTP control message, typically small, that must be immediately acknowledged upon reception. They carry no payload data and are used as probes to actively monitor unused paths, where no retransmissions are available for PFD. We use heartbeats as probes in our algorithm. This is how it works:

- PMR consecutive retransmissions no longer trigger a path failure detection.
- Starting with the first retransmission, a succession of heartbeats will be issued to actively monitor the path state. There is no need for these heartbeats to be reliable, so they are not going to be scheduled for retransmission.
- The heartbeats will be issued at a fixed pace of:

$$T_{\mathrm{hb}} = \frac{D_{\max}}{n} \tag{1}$$

  Where $D_{\max}$ is the maximum admissible duration of the failover and n is the number of probes to send. $D_{\max}$ can be an application imposition or a well crafted value based on the mean duration of the error bursts present in the system. n can be chosen for a certain probability of false detections.
- The lack of acknowledgments to the heartbeats must not trigger the exponential backoff of the RTT estimation.
- To prevent congestion, there must be room left on the congestion window to issue the sequence of heartbeats.
- If one of these heartbeats is acknowledged, there hasn't been a path failure and the heart beating stops. If no heartbeat has been acknowledged after the time T_{hb}, a path failure is detected and the failover mechanism is triggered.

Note that the sequence of heartbeats is not a heartbeat and its retransmissions, but a sequence of n equally spaced, independent heartbeats. This could lead to loss of stability as [6] suggest and is one of our directions for future work.

For the experiment we used two Linux hosts running a modified version of lksctp[3] that met the above requirements.

[3]Linux Kernel SCTP implementation for a 2.6.18 Linux kernel

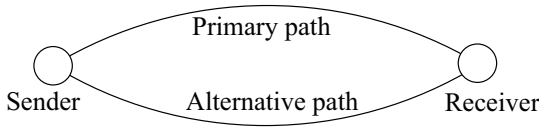

Fig. 1. Layout of the experiment

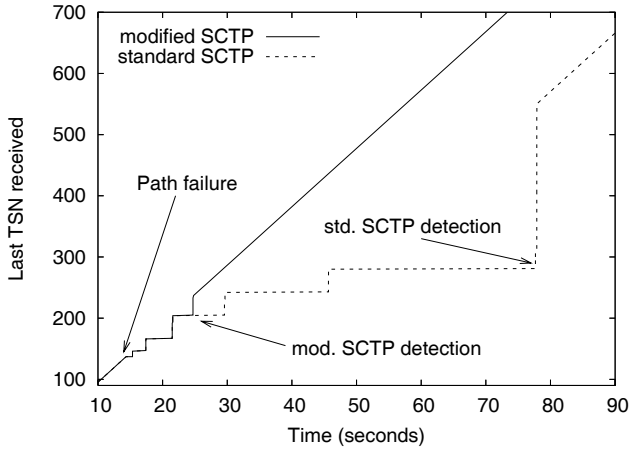

Fig. 2. This figure shows the last received sequence number as a function of time with the standard SCTP driver and the modified one.

Figure 1 shows the layout of the experiment. Sender and receiver had two 100baseT Ethernet NICs each. The alternative path was a direct connection between sender and receiver. The primary path had an Ethernet hub. We simulate path failures by manually unplugging the Ethernet cable of the receiver from the hub side. There were no other traffic on the paths. The measured delay on both paths was 1 ms.

Figure 2 compares the received TSNs when a sender is using a plain lksctp kernel and a modified one. The sender issued 40 data bytes each 100 ms to the receiver[4]. A path failure was manually forced at second 15. $D_{\max}$ was set to 10 seconds and n was set to an extremely conservative value of 10. The standard SCTP needs 63 seconds to failover while the modified one took $D_{\max}$ (10 seconds). The results show how easy is it to tune a decoupled PFD algorithm without modifying the congestion control dynamics.

IV. Related work

There have been a number of different approaches to reduce the failover time of SCTP; the following texts have influenced the work presented in this letter. First there are some works that try to tune different SCTP parameters for a better performance:

Reduce PMR: This is one of the most effective ways of reducing the handover time. But it increases the chance of false detection. In [7] the authors state that for loss rates between 1% and 8%, values of PMR between 0 and 5 are effective to detect path failures. It is also proved that a PMR of 0 improves the goodput over $\mathrm{PMR} > 0$. They found that spurious failovers don't degrade the performance in their scenario. Their conclusions are very interesting in scenarios

[4]This low data rate empties the congestion window slowly, which allows for a clearer representation of the retransmissions at figure 2

where redundant paths are designed, deployed and controlled by the runner of the application, where (frequent) switching is not a problem. Networks with higher error rates or error bursts (like wireless networks) or scenarios with redundant public paths, like the internet, don't fit very well in their results.

Reduce the acknowledgment delay: Acknowledgments are not sent immediately, they are delayed to reduce traffic overhead. In [8] there is a study on reducing the acknowledgment delay and its effect on the overall handover time. The conclusion is that the effect of reducing this delay is greater on slower networks. On the best case scenario, with no delay, the handover time improves in less than a 10%, and the acknowledgment traffic is doubled.

There are other approaches that try to adapt PFD to certain applications by modifying the standard behavior of SCTP:

On [9] there is an analysis of several retransmission algorithms. Their experiments show that sending a heartbeat after the first retransmission halves the handover time, because the unacknowledged heartbeat contributes to the error counter of the path.

The authors of [10] set the path failure mechanism to choose the path with the lowest RTT. The proposal is interesting and novel and their experiments give some impressive results. There is still some room to improve the RTT estimations, by measurement filtering and the use of hysteresis. This approach eliminates the meaning of a primary path, since the path to use will always be the one with less RTT. This can be harmful for scenarios where there is truly a primary path, whether by cost, policies...

On [11], the authors face the problem of using SCTP in *concurrent multipath transfer* scenarios, and they propose active monitoring with heartbeats on the first retransmission. This is nearly identical to our active monitoring, except they use exponential backoff to pace theirs probes.

None of these works clearly states that PFD should be independent of other protocol mechanisms, but [10] and [11] point in that direction.

On [12], the authors present how SCTP can stall in certain circumstances. While that problem is not directly related to the topics of this letter, they state that the fundamental cause of the problem is the coupling between acknowledgements and path monitoring. Their conclusions served as an inspiration for this letter.

V. Conclusion

Tuning SCTP to improve its failover time is difficult as it implies tradeoffs. These tradeoffs come from the tight coupling between congestion control and path failure detection: SCTP uses retransmissions as probes for path failure detection and congestion control forces an exponential backoff pacing of retransmissions. Congestion control constrains path failure detection.

We propose to stop using retransmissions as probes for path failure detection. Instead we provide a simple algorithm based on active path monitoring using non-reliable periodic heartbeats. This algorithm has configurable parameters to adapt it to different application requirements.

The limits of this approach depend on the feasibility of the application requirements for the available networks not on the congestion control limitations. We show the results from a simple experiment that compares our approach with the standard SCTP.

Other algorithms can be used to detect path failures. Future work is oriented towards exploring more advanced algorithms and how to make them fulfill different application requirements and stability issues.

References

[1] R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Proposed Standard), Sept. 2007.

[2] V. Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium*, Stanford, CA, August, 1988, vol. 18, 4, pp. 314–329, 1988. [Online]. Available: http://citeseer.ist.psu.edu/article/jacobson88congestion.html

[3] V. Paxson and M. Allman, "Computing TCP's retransmission timer," RFC 2988 (proposed standard), Nov. 2000.

[4] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," *ACM Trans. Computer Syst.*, vol. 9, pp. 2–7, 1987.

[5] L. Budzisz, R. Ferrus, K.-J. Grinnemo, A. Brunstrom, and F. Casadevall, "An analytical estimation of the failover time in SCTP multihoming scenarios," in *Proc. IEEE Wireless Communications and Networking Conference (WCNC 2007)*.

[6] D. E. Comer, "13.18 Karn's Algorithm And Timer Backoff," in *Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architectures, Fourth Edition*. Upper Saddle River, NJ: Prentice Hall PTR, 2000, p. 229.

[7] J. Caro, A. L., P. D. Amer, and R. R. Stewart, "End-to-end failover thresholds for transport layer multihoming," in *Proc. IEEE Military Comunications Conference 2004 (MILCOM)*, vol. 1, pp. 99–105, vol. 1, 2004.

[8] J. Eklund and A. Brunstrom, "Impact of sack delay and link delay on failover performance in SCTP," in *Commun. and Computer Networks*, pp. 69–74, 2006.

[9] J. Caro, P. Amer, and R. Stewart, "Retransmission schemes for end-to-end failover with transport layer multihoming," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM '04)*, vol. 3, pp. 1341–1347, vol. 3, 2004.

[10] A. Kelly, G. Muntean, P. Perry, and J. Murphy, "Delay-centric handover in sctp over WLAN," *Trans. Automatic Control and Computer Science*, vol. 49, 2004.

[11] J. R. Iyengar and R. R. Amer, P. D. Stewart, "Concurrent multipath transfer using sctp multihoming over independent end-to-end paths," *IEEE/ACM Trans. Networking*, vol. 14, no. 5, pp. 951–964, 2006.

[12] J. Noonan, P. Perry, S. Murphy, and J. Murphy, "Stall and path monitoring issues in SCTP," in *Proc. INFOCOM*, 2006.