An Application Demonstration of the Reliable Server Pooling Framework

Thomas Dreibholz, Erwin P. Rathgeb
University of Essen
Institute for Experimental Mathematics
Ellernstraße 29, D-45326 Essen, Germany
{dreibh, rathgeb}@exp-math.uni-essen.de
Tel: +49 201 183-{7637, 7670}

Abstract

The convergence of classical PSTN and IP networks requires the transport of SS7 signaling over IP. Since SS7 has very strict availability requirements for the signaling components, redundancy is mandatory. The goal of the IETF RSerPool working group is to define a lightweight, flexible and real-time redundancy concept to fulfill the availability requirements of SS7: Reliable Server Pooling (RSerPool). RSerPool is currently under standardization, its functionality and improvement are subject of our research.

As part of our RSerPool research, we have created an implementation prototype together with an example application. In our proposed demo, we will explain the fundamental protocol mechanisms of RSerPool and demonstrate its behavior and the impacts of component failures to an example application.

1 What is Reliable Server Pooling?

The convergence of classical circuit-switched networks (i.e. PSTN/ISDN) and data networks (i.e. IP-based) is rapidly progressing. This implies that PSTN signaling via the SS7 protocol is transported over IP networks. Since SS7 signaling networks offer a very high degree of availability (e.g. at most 10 minutes downtime per year for any signaling relationship between two signaling endpoints; for more information see [9]), all links and components of the network devices must be redundant. When transporting signaling over IP networks, such redundancy concepts also have to be applied to achieve the required availability. Link redundancy in IP networks is supported using the Stream Control Transmission Protocol (SCTP [12, 10]); redundancy of network device components is supported by the SGP/ASP (signaling gateway process/application server process) concept. However, this concept has some limitations: no support of dynamic addition and removal of components, limited ways of server selection, no specific failover procedures and inconsistent application to different SS7 adaptation layers.

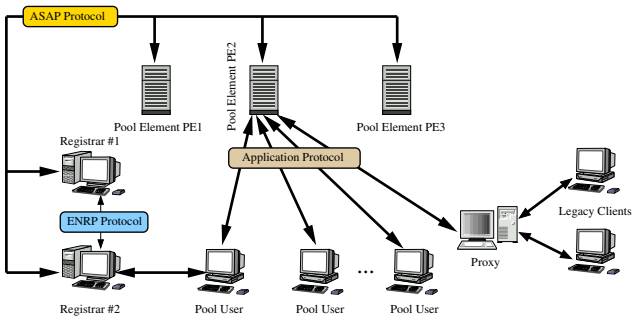

Figure 1. An Overview of the RSerPool Architecture

To cope with the challenge of creating a unified, lightweight, real-time and flexible redundancy solution, the IETF Reliable Server Pooling Working Group has been founded. An overview of their concept Reliable Server Pooling (RSerPool), which is currently in the standardization process and described by several Internet Drafts, is shown in figure 1. Redundant servers providing the same service belong to a so called *server pool*, identified by a unique ID called *pool handle* within the set of all server pools, the so called *handlespace*. A server in a pool is called *pool element* (PE) of its pool. The handlespace is managed by redundant *registrars* (PRs), who synchronize their view of the handlespace using the Endpoint haNdlespace Redundancy Protocol (ENRP [16]). PRs can announce themselves using multicast announcements, i.e. it is not necessary (but possible) to pre-configure PR addresses to the components described in the following.

PEs can register to a pool of the handlespace at an arbitrary PR using the Aggregate Server Access Protocol

(ASAP [13]). A suitable PR is automatically found by listening to PR multicast announcements or by static configuration. The PR chosen by the PE for registration monitors the PE using SCTP heartbeats and ASAP keep-alives; the frequency of monitoring messages depends on the provided service's availability requirements. When a PE becomes unavailable, it is immediately removed from the handlespace. A PE can also intentionally deregister from the handlespace by an ASAP deregistration. PR failures are handled by requiring PEs to re-register regularly (and therefore choosing a new PR when necessary) and by PRs monitoring their peer PRs and invoking an ENRP takeover procedure in case of failure.

When a client requests a service from a pool, it asks an arbitrary PR to translate the pool handle to a list of PE transport addresses selected by the pool's selection policy (*pool policy*), e.g round robin or least used (see [14, 7] for additional standardized policies). Then, it selects again one PE and establishes a transport connection to this PE using the application's protocol. The client then becomes a so called *pool user* (PU) of the PE's pool. In case of PE failure, the procedure is repeated to establish a connection to a new PE. Optionally, a PU can report a PE failure to a PR, which may decide to remove this PE from the handlespace.

RSerPool supports optional client-based state synchronization [3] for failover. That is, a PE can provide its current state as *state cookie* to the PU. When a failover to a new PE is made, the PU can transmit this state cookie to the new PE, which can then restore this state. RSerPool is not restricted to client-based state synchronization, any other application-specific failover procedure can be used as well by the application layer itself.

The lightweight, real-time and flexible architecture of RSerPool is not only usable for SS7-based telephony signaling. Other application scenarios include SIP [2], mobility management [6] and the management of distributed computing pools [8, 17].

Finally, load balancing using RSerPool is currently under discussion by the IETF RSerPool Working Group: due to its flexible server selection policies and pool management functionalities, it has many similarities to load balancer protocols. A very common application for such load balancing systems is to distribute HTTP requests in web server farms. There is an ongoing effort to merge both the RSerPool framework and the Server/Application State Protocol (SASP [1], a contribution of IBM) for load balancers into one common architecture for highly-available server pool management and load distribution.

2 Our Demonstration System

Currently, there are only two existing implementations of RSerPool: a closed source version by Motorola [15] and the

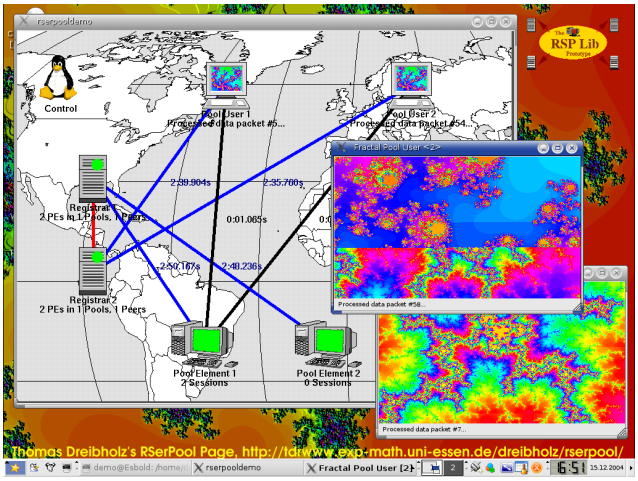

Figure 2. Our RSerPool Application Demo

authors' own GPL-licensed Open Source prototype *rsplib*. The latter implementation has been created by us as part of our RSerPool research and to verify the results of our simulation model [7, 4, 5] in real-life scenarios. It is a complete implementation prototype of the RSerPool framework [11, 8], together with a vivid example application. In this application, the PEs provide a computation service for fractal graphics and can dynamically join and leave the pool. At least two PRs manage the handlespace the pool belongs to.

The computation service provided by the pool is requested by PUs, which continuously show the calculation results on screen; i.e. any service disturbances are immediately visible. Our calculation service example therefore becomes representative for various kinds of RSerPool application scenarios, due to its similar requirements on failure and fail-over handling:

- It must be possible to continue the interrupted service on a new PE.
- A failover should be as quick as possible. Otherwise, the service user would be annoyed.

To achieve the first requirement, client-based state synchronization [3] is applied as it is provided [13] by RSerPool. The second requirement, real-time failover behavior, is a design criteria of RSerPool. Our PU application can vividly display that RSerPool achieves both requirements of the service.

The GUI-based control component, shown in figure 2, permanently shows each component's status and the protocol associations between the devices. Furthermore, it is able to interactively start and stop – either by clean shutdown or by hard shut-off like in case of a failure – arbitrary components.

It should be stated again that our demo system is not a simulation, it consists of our real, Linux-based RSerPool implementation and real application programs.

3 Our Proposed Demo

In our demo, we first explain and demonstrate the fundamental RSerPool protocol mechanisms for dynamic, completely auto-configuring pool management and the mapping of PU requests to PEs by policy-based server selection. Here, we can demonstrate the impact of the various PE selection strategies (pool policies) on load balancing performance.

In the following, we will show the effects of PE and PR shut-downs and failures: while the behavior of the RSerPool framework can be monitored in the GUI's display, the impacts of PEs shut down or failed on the pool's provided service can be observed in real-time by having a look at the PU's result display progress. On a separate screen the SCTP and RSerPool messages can be captured and decoded to provide additional insight into the protocol mechanisms.

Thus, we will show that RSerPool is not only able to provide auto-configuring management of dynamic server pools but also ensures quick failover and service continuation in case of component failures.

For our demo presentation, we only require space and power outlets for three laptop computers.

References

[1] A. Bivens. Server/Application State Protocol v1. Internet-Draft Version 01, IETF, Individual submission, Oct 2004. draft-bivens-sasp-01.txt, work in progress.

[2] P. Conrad, A. Jungmaier, C. Ross, W.-C. Sim, and M. Tüxen. Reliable IP Telephony Applications with SIP using RSerPool. In *Proceedings of the SCI 2002, Mobile/Wireless Computing and Communication Systems II*, volume X, Orlando/U.S.A., Jul 2002.

[3] T. Dreibholz. An efficient approach for state sharing in server pools. In *Proceedings of the 27th Local Computer Networks Conference*, Tampa, Florida/U.S.A., Oct 2002.

[4] T. Dreibholz. An Overview of the Reliable Server Pooling Architecture. In *Proceedings of the International Conference on Network Protocols 2004*, Berlin/Germany, Oct 2004.

[5] T. Dreibholz. Policy Management in the Reliable Server Pooling Architecture. In *Proceedings of the Multi-Service Networks Conference 2004*, Abingdon, Oxfordshire/United Kingdom, Jul 2004.

[6] T. Dreibholz, A. Jungmaier, and M. Tüxen. A new Scheme for IP-based Internet Mobility. In *Proceedings of the 28th Local Computer Networks Conference*, Königswinter/Germany, Nov 2003.

[7] T. Dreibholz, E. P. Rathgeb, and M. Tüxen. Load Distribution Performance of the Reliable Server Pooling Framework. In *Proceedings of the International Conference on Networking 2005*, Saint Gilles Les Bains/Reunion Island, Apr 2005.

[8] T. Dreibolz and M. Tüxen. High availability using reliable server pooling. In *Proceedings of the Linux Conference Australia 2003*, Perth/Australia, Jan 2003.

[9] K. D. Gradischnig and M. Tüxen. Signaling transport over IP-based networks using IETF standards. In *Proceedings of the 3rd International Workshop on the design of Reliable Communication Networks*, pages 168–174, Budapest, Hungary, 2001.

[10] A. Jungmaier, M. Schopp, and M. Tüxen. Performance Evaluation of the Stream Control Transmission Protocol. In *Proceedings of the IEEE Conference on High Performance Switching and Routing*, Heidelberg/Germany, June 2000.

[11] Thomas Dreibholz's RSerPool Page. http://tdrwww.exp-math.uni-essen.de/dreibholz/rserpool.

[12] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. RFC 2960, IETF, Oct 2000.

[13] R. Stewart, Q. Xie, M. Stillman, and M. Tüxen. Aggregate Server Access Protcol (ASAP). Internet-Draft Version 10, IETF, RSerPool WG, Oct 2004. draft-ietf-rserpool-asap-10.txt, work in progress.

[14] M. Tüxen and T. Dreibholz. Reliable Server Pooling Policies. Internet-Draft Version 00, IETF, RSerPool WG, Oct 2004. draft-ietf-rserpool-policies-00.txt, work in progress.

[15] Q. Xie. Private communication at the 60th IETF meeting, San Diego/California, U.S.A., August 2004.

[16] Q. Xie, R. Stewart, and M. Stillman. Endpoint Name Resolution Protcol (ENRP). Internet-Draft Version 10, IETF, RSerPool WG, Oct 2004. draft-ietf-rserpool-enrp-10.txt, work in progress.

[17] Y. Zhang. Distributed Computing mit Reliable Server Pooling. Masters thesis, Universität Essen, Institut für Experimentelle Mathematik, Apr 2004.