Evaluation of Attack Countermeasures to Improve the DoS Robustness of RSerPool Systems by Simulations and Measurements

Xing Zhou[1], Thomas Dreibholz[2], Wencai Du[1], Erwin P. Rathgeb[2]

[1] Hainan University, College of Information Science and Technology
Renmin Avenue 58, 570228 Haikou, Hainan, China
{zhouxing,wencai}@hainu.edu.cn
[2] University of Duisburg-Essen, Institute for Experimental Mathematics
Ellernstrasse 29, 45326 Essen, Germany
{dreibh,rathgeb}@iem.uni-due.de

Abstract The Reliable Server Pooling (RSerPool) architecture is the IETF's new standard for a lightweight server redundancy and session failover framework to support availability-critical applications. RSerPool combines the ideas from different research areas into a single, resource-efficient and unified architecture. While there have already been a number of research papers on its performance in general, the robustness against intentional attacks has not been intensively addressed yet. In particular, there have not been any analyses for real setups.
Therefore, the goal of this paper is to provide a robustness analysis in order to outline the attack bandwidth which is necessary for a significant impact on RSerPool-based services. This analysis is based on lab measurements – using a real RSerPool system setup – as well as on measurements for comparison and validation. Furthermore, we present and evaluate countermeasure approaches to significantly reduce the impact of attacks.[1]
Keywords: Reliable Server Pooling, Security, Attacks, Denial of Service, Robustness, Performance Analysis

1 Introduction and Scope

Reliable Server Pooling (RSerPool, see [15]) denotes the IETF's new standard for a generic, application-independent server pool [7] and session management [3] framework. While there have already been a number of publications on the performance of RSerPool for load balancing [3, 4, 11, 22–24] and server failure handling [8], there has been very little research on its security and attack robustness. Until now, only basic concepts to avoid flooding the pool management with misinformation have been analysed by simulations in [9, 16]. The underlying transport protocol SCTP[2] already provides protection against blind flooding attacks [20] and the RFC [19] of RSerPool mandatorily requires applying mechanisms like TLS [14] or IPSEC [1] in order to ensure

[1] Funded by the State Administration of Foreign Experts Affairs, P. R. China (funding number 20084600036) and the German Research Foundation (Deutsche Forschungsgemeinschaft).
[2] Stream Control Transmission Protocol, see [17].

authenticity, integrity and confidentiality. Nevertheless, these techniques are still insufficient: in a distributed system, there is always a chance that an attacker compromises a legitimate component (e.g. by exploiting a software bug) and obtains the private key. It is therefore important to analyse the behaviour of the RSerPool protocols under attack situations.

The goal of this paper is to analyse the attack robustness of the RSerPool architecture by simulations using our RSerPool simulation model RSPSIM [6] as well as measurements in a lab setup based on our RSerPool implementation RSPLIB [3] – which is also the IETF's reference implementation, see [15, chapter 5] – by first showing the impact of different attack scenarios on the application performance. Using these analyses as a baseline performance level, we will present techniques to efficiently reduce the impact of such attacks.

2 The RSerPool Architecture

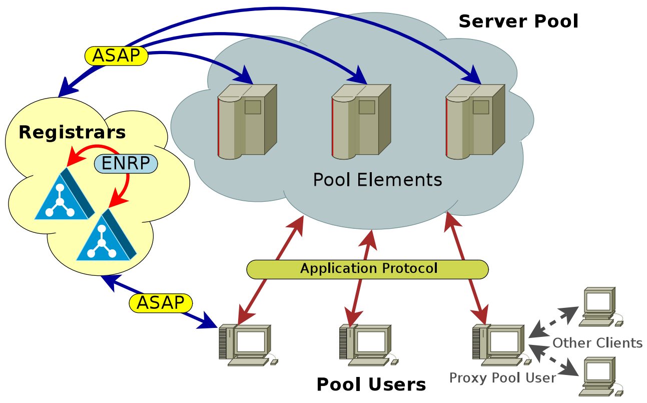

Figure 1. The RSerPool Architecture

Figure 1 illustrates the RSerPool architecture [3,15] which consists of three types of components: servers of a pool are called *pool elements* (PE), a client is denoted as *pool user* (PU). The *handlespace* – which is the set of all pools – is managed by redundant *pool registrars* (PR). Within the handlespace, each pool is identified by a unique *pool handle* (PH).

PRs of an *operation scope* synchronize their view of the handlespace by using the Endpoint haNdlespace Redundancy Protocol (ENRP) [21], transported via SCTP [13] and secured e.g. by TLS [14] or IPSEC [1]. Unlike Grid Computing [12], an operation scope is restricted to a single administrative domain. That is, all of its components are under the control of the same authority (e.g. a company or an organization). This property results in a small management overhead [7], which also allows for RSerPool usage on devices providing only limited memory and CPU resources (e.g. embedded systems like routers). Nevertheless, PEs may be distributed globally to continue their service even in case of localized disasters [5].

PEs choose an arbitrary PR of the operation scope to register into a pool by using the Aggregate Server Access Protocol (ASAP) [18], again transported via SCTP and using TLS or IPSEC. Within its pool, a PE is characterized by its PE ID, which is a randomly chosen 32-bit number. Upon registration at a PR, the chosen PR becomes the Home-PR (PR-H) of the newly registered PE. A PR-H is responsible for monitoring its PEs' availability by keep-alive messages (to be acknowledged by the PE within a given timeout) and propagates the information about its PEs to the other PRs of the operation scope via ENRP updates. PEs re-register regularly (in an interval denoted as *registration lifetime*) and for information updates.

In order to access the service of a pool given by its PH, a PU requests a PE selection from an arbitrary PR of the operation scope, again using ASAP. The PR selects the requested list of PE identities by applying a pool-specific selection rule, called *pool policy*. RSerPool supports two classes of load distribution policies: non-adaptive and adaptive algorithms [4]. While adaptive strategies base their assignment decisions on the current status of the processing elements (which of course requires up-to-date states), non-adaptive algorithms do not need such data. A basic set of adaptive and non-adaptive pool policies is defined in [10]. Relevant for this paper are the non-adaptive policies Round Robin (RR) and Random (RAND) as well as the adaptive policies Least Used (LU) and Least Used with Degradation (LUD). LU selects the least-used PE, according to up-to-date application-specific load information. Round robin selection is applied among multiple least-loaded PEs. LUD [24] furthermore introduces a *load decrement* constant which is added to the actual load each time a PE is selected. This mechanism compensates inaccurate load states due to delayed updates. An update resets the load to the actual load value again.

PUs may report unreachable PEs to a PR by using an ASAP Endpoint Unreachable message. A PR locally counts these reports for each PE and when reaching the threshold $\mathrm{MaxBadPEReports}$ [8] (default is 3 [18]), the PR may decide to remove the PE from the handlespace. The counter of a PE is reset upon its re-registration.

3 Quantifying an RSerPool System

For our quantitative performance analysis, we use the application model from [3]: the service provider side of an RSerPool system consists of a pool of PEs. Each PE has a request handling *capacity*, which we define in the abstract unit of calculations per second[3]. Each request consumes a certain number of calculations; we call this number *request size*. A PE can handle multiple requests simultaneously – in a processor sharing mode as provided by multitasking operating systems.

On the service user side, there is a set of PUs. The number of PUs can be given by the ratio between PUs and PEs (*PU:PE ratio*), which defines the parallelism of the request handling. Each PU generates a new request in an interval denoted as *request interval*. The requests are queued and sequentially assigned to PEs.

[3] An application-specific view of capacity may be mapped to this definition, e.g. CPU cycles, harddisk space, bandwidth share or memory usage.

The total delay for handling a request d_{Handling} is defined as the sum of queuing delay d_{Queuing}, startup delay d_{Startup} (dequeuing until reception of acceptance acknowledgement) and processing time $d_{\text{Processing}}$ (acceptance until finish):

$$d_{\text{Handling}} = d_{\text{Queuing}} + d_{\text{Startup}} + d_{\text{Processing}}. \tag{1}$$

That is, d_{Handling} not only incorporates the time required for processing the request, but also the latencies of queuing, server selection and message transport. The user-side performance metric is the *handling speed*, which is defined as:

$$\text{HandlingSpeed} = \frac{\text{RequestSize}}{d_{\text{Handling}}}.$$

For convenience reasons, the handling speed (in calculations/s) is represented in $\%$ of the average PE capacity.

Using the definitions above, it is possible to delineate the average system utilization (for a pool of NumPEs servers and a total pool capacity of PoolCapacity) as:

$$\text{SystemUtilization} = \text{NumPEs} * \text{puToPERatio} * \frac{\frac{\text{RequestSize}}{\text{RequestInterval}}}{\text{PoolCapacity}}. \tag{2}$$

Obviously, the provider-side performance metric is the system utilization, since only utilized servers gain revenue. In practise, a well-designed client/server system is dimensioned for a certain *target system utilization* of e.g. 50%. That is, by setting any two of the parameters (PU:PE ratio, request interval and request size), the value of the third one can be calculated using equation 2 (see [3] for detailed examples).

4 System Setup

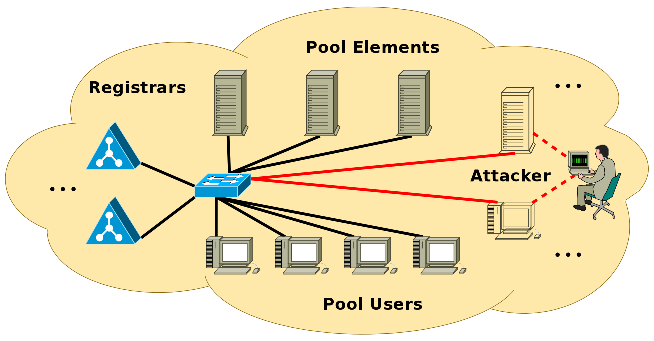

Figure 2. The System Setup

For our performance analysis, we have used the OMNeT++-based RSerPool simulation model RSPSIM [4, 6] as well as the implementation RSPLIB [3, 5] for measurements in a lab setup. Both – simulation model and implementation – contain the protocols ASAP [18] and ENRP [21], a PR module, an attacker module and PE as well as PU modules for the request handling scenario defined in section 3.

Unless otherwise specified, the basic simulation and measurement setup – which is also presented in figure 2 – uses the following parameter settings:

- The target system utilization is 50%. Request size and request interval are randomized using a negative exponential distribution (in order to provide a generic and application-independent analysis [3, 4]). There are 10 PEs; each one provides a capacity of 10^6 calculations/s.
- A PU:PE ratio of 3 is used (i.e. a non-critical setting as explained in [4]).
- We use request size:PE capacity setting of 10; i.e. being processed exclusively, the average processing takes 10s – see also [4].
- There is a single PR only, since we do not examine PR failure scenarios here (see [4] for the impact of multiple PRs). PEs re-register every 30s (registration lifetime) and on every load change of the adaptive LU and LUD policies.
- $\mathrm{MaxBadPEReports}$ is set to 3 (default value defined in [18]). A PU sends an Endpoint Unreachable if a contacted PE fails to respond within 10s (see also [8]).
- The system is attacked by a single attacker node.
- For the simulation, the simulated real-time is 120min; each simulation run is repeated at least 24 times with a different seed in order to achieve statistical accuracy.
- Each measurement run takes 15min; each run is repeated at least 3 times.

GNU R [6] is used for the statistical post-processing of the results. Each resulting plot shows the average values and their 95% confidence intervals.

5 The Impact of an Attacker

Attack targets of RSerPool systems are the PRs, PEs and PUs. Due to the restriction of RSerPool to a single administrative domain, a protection of the small number of PRs is assumed to be feasible [9, 16]. Instead, the most likely attack targets are the PEs and PUs. These components are significantly more numerous [3] and may be distributed over a larger, less controllable area [5]. For that reason, ASAP-based attacks are in the focus of our study. Initially, we will show that – without any protection mechanisms – even a *single* compromised PE or PU can already cause a Denial of Service (DoS).

5.1 An Attacker Masquerading as Pool Element

The goal of an attacker being able to perform PE registrations is clearly to perform as many fake registrations as possible. That is, each registration request simply has to contain a new (random) PE ID. The policy parameters may be set appropriately, i.e. a load of 0% (LU, LUD) and a load increment of 0% (LUD), to get the fake PE selected as frequently as possible. The underlying SCTP protocol [17, 20] itself already prevents simple address spoofing: each network-layer address under which a PE is registered must be part of the SCTP association between PE and PR. The ASAP protocol [18] requires the addresses to be validated by SCTP. However, maintaining a registration

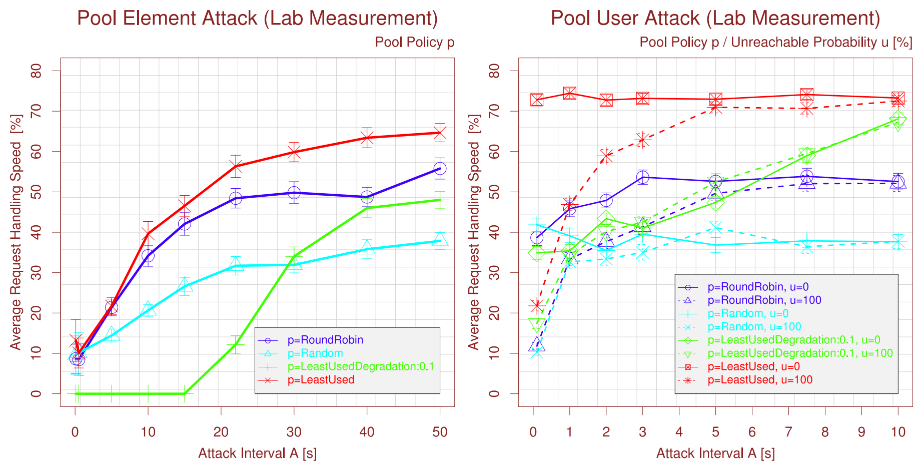

Figure 3. The Impact of a PE/PU-Based Attacks without Countermeasures

association with the PE and silently dropping all incoming PU requests is already sufficient for an attacker.

The left-hand side of figure 3 shows the impact of varying the attack interval A (i.e. the time between two fake registrations) on the handling speed in the lab setup. Clearly, even an interval of 10s (i.e. one registration packet per 10s) is already sufficient to significantly decrease the performance. For the LUD policy [24], this already leads to a full DoS: an unloaded PE whose load does not increase when accepting a new request seems to be the most useful choice for each PU. That is, it leads to exclusively choosing fake PEs. Using smaller settings of A (e.g. here: 0.1s) a DoS is also reached for the other policies.

The corresponding simulation results show an analogous behaviour. Since we have presented such results already in [9, 16], they have been omitted here.

5.2 An Attacker Masquerading as Pool User

PUs are the other means of ASAP-based attacks, especially their handle resolution requests and unreachability reports. Obviously, an attack scenario would be to flood the PR with handle resolution requests. However, the server selection as part of the handlespace management is very efficiently realizable [7] – but it is a fallacy to assume that simply performing some handle resolutions (without actually contacting any selected PE) cannot affect the service quality: on the right-hand side of figure 3, the impact of a handle resolution attack on the performance is presented for varying the attack interval A (i.e. the delay between two handle resolution requests) in the lab setup. For selected PE entries, an unreachability report (see section 2) is sent with probability u.

Even for u=0%, there is an impact on the performance of RR, due to the "stateful" [4] operation of RR: the round robin pointer is advanced by the selection procedure itself (i.e. without actually using a PE), causing the selection of less appropriate

PEs. Similarly, the load value is increased on each selection for LUD. In contrast, LU and RAND are "stateless" and therefore not affected by this attack. The impact of also reporting all PEs as being unreachable (here: u=100%) is disastrous: PEs are kicked out of the handlespace, and the handling speed quickly sinks and leads – here at about A=0.1s (i.e. only 10 reports/s) to a DoS. Since analogous simulation results have been discussed by us in [9, 16], we omit them here.

6 Applying Attack Countermeasures

Clearly, even a single attacker with a small attack bandwidth (i.e. a few messages/s) can achieve a complete DoS. Therefore, we discuss and analyse possible countermeasures for the PE and PU-based threats in the following section.

6.1 Countermeasures Against Pool Element Attacks

The key problem of the PE-based threat shown in subsection 5.1 is the attacker's ability to create a new fake PE with each of its registration messages. Only a few messages per second (i.e. even a modem connection) are sufficient to degrade the service. Therefore, an effective countermeasure is to restrict the number of PE registrations that a single PE identity is allowed to create. However, in order to retain the "lightweight" [7] property of RSerPool and to avoid synchronizing such numbers among PRs, we have put forward a new approach and introduce the concept of a *registration authorization ticket* [9] consisting of:

1. the pools's PH and a fixed PE ID,
2. minimum/maximum policy information settings (e.g. a lower limit for the load decrement of LUD) and
3. a signature by a trusted authority (to be explained below).

This ticket, provided by a PE to its PR-H as part of the ASAP registration, can be easily verified by checking its signature. Then, if it is valid, it is only necessary to ensure that the PE's policy settings are within the valid range specified in the ticket. An attacker stealing the identity of a real PE would only be able to masquerade as *this* specific PE. A PR only has to verify the authorization ticket. No protocol change or synchronization among the PRs is necessary. The additional runtime required is in $O(1)$. Clearly, the need for a trusted authority (e.g. a Kerberos service) adds an infrastructure requirement. However, since an operation scope is restricted to a single administrative domain (see section 2), this is feasible at reasonable costs.

To demonstrate the effectiveness of our countermeasure approach, figure 4 presents the handling speed results for an attack interval of A=0.1s per attacker for varying the number of attackers α. As shown in subsection 5.1, A=0.1s has already lead to a full DoS with only a *single* attacker.

Obviously, our countermeasure is quite effective: even for α=10, the handling speed only halves at most – but the service still stays operational and the attack impact is not even close to a DoS. Note, that α=10 here means that the attacker had successfully compromised as many nodes for obtaining registration authorization tickets as there are

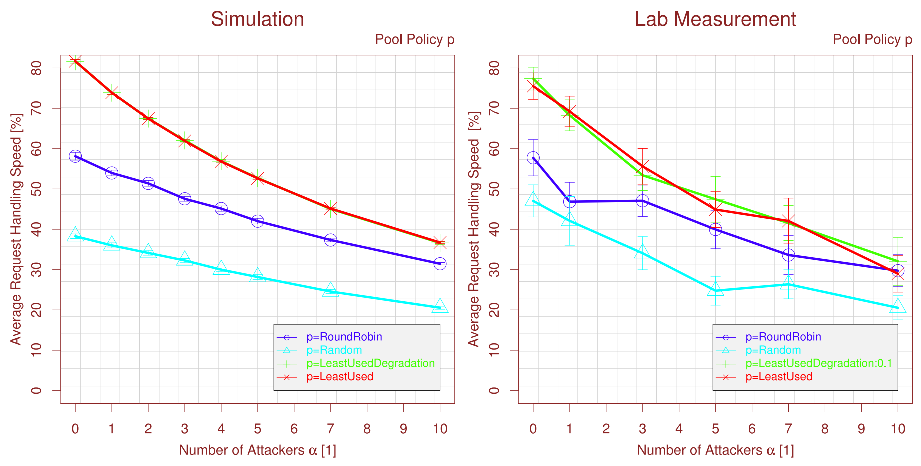

Figure 4. Applying Countermeasures Against Pool-Element-Based Attacks

"real" PEs in the pool – which is assumed to be quite difficult in a controlled RSerPool operation scope (i.e. a single administrative domain, see section 2). The lab measurement results correspond to the simulation results, i.e. the mechanism also works effectively in a real setup. Note, that the slightly different handling speed levels of the lab measurements are caused by node and SCTP association setup latencies, which are – due to their complexity – not incorporated into the simulation model. Nevertheless, the obtained tendency of the results is clearly observable.

6.2 Countermeasures Against Pool User Attacks

The key threat of the handle resolution/failure report attack shown in subsection 5.2 is a PU's ability to impeach PEs – even for $\mathrm{MaxBadPEReport}>1$. Our basic countermeasure idea is therefore to avoid counting multiple reports from the same PU. As for the PEs, it is necessary to introduce a PU identification which is certified by a trusted authority and can be verified by a PR (see subsection 6.1). Then, a PR simply has to memorize (described later) the PH for which a certain PU has reported unreachable PEs and to ignore multiple reports for the same pool. Since the unreachability count for each PE is a PR-local variable, no synchronization among PRs is necessary. That is, an attacker cannot cause harm by just sending its reports for the same PE to different PRs.

Instead of storing each reported PE (which could be exploited by sending a large amount of random IDs), we use a hash-based approach for a per-PU message blackboard: the function $\varPsi$ maps a PE's PH into a bucket:

$$\varPsi(\mathrm{PH}) = h(\mathrm{PH})\ \textsf{MOD}\ \mathrm{Buckets}.$$

h denotes an appropriate hash function: an attacker may not easily guess its behaviour. This property is provided by so called universal hash functions [2], which are – unlike cryptographic hash functions (e.g. MD5, SHA-256) also efficiently computable.

Each bucket contains the time stamps of the latest up to $\mathrm{MaxEntries}$ Endpoint Unreachables for the corresponding bucket. Then, the report rate can be calculated as:

$$\mathrm{Rate} = \frac{\mathrm{NumberOfTimeStamps}}{\mathrm{TimeStamp_{Last}} - \mathrm{TimeStamp_{First}}}. \tag{3}$$

Upon reception of an Endpoint Unreachable, it simply updates the reported PE's corresponding bucket entry. If the rate in equation 3 exceeds the configured threshold $\mathrm{MaxEURate}$ (Maximum Endpoint Unreachable Rate), the report is silently ignored. The effort for this operation is in $O(1)$, as well as the required per-PU storage space.

In a similar way, the same hash-based approach can be applied for handle resolutions with the corresponding threshold $\mathrm{MaxHRRate}$ (Maximum Handle Resolution Rate). However, unlike simply ignoring the request, the PR replies with an empty list – which means for the PU that the pool is currently empty.

Instead of specifying a fixed threshold, an alternative approach is presented in [16] by applying statistical anomaly detection: the behaviour of the majority of nodes is assumed to be "normal". Differing behaviour – which is necessary for an attack to be effective – is denoted as anomaly. However, this approach can – by definition – only detect attackers if their number is less than the number of legitimate components. Furthermore, obtaining the "normal" behaviour is more resource-intensive than simple thresholds.

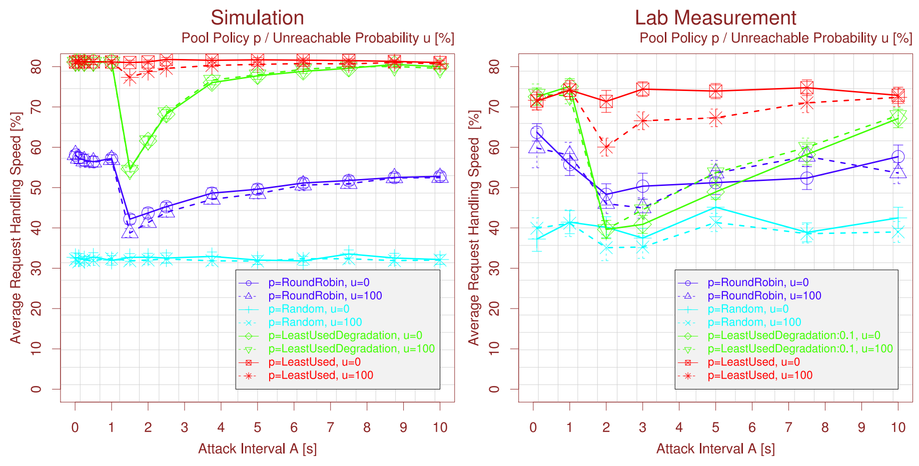

Figure 5. Applying Countermeasures Against a Pool-User-Based Attack

The effectiveness of our approach for $\mathrm{MaxHRRate}$=1 (which is 60 times more than the application's actual handle resolution rate) and $\mathrm{MaxEURate}$=1 (which is by orders of magnitude higher than a real pool's PE failure rate) is shown by figure 5 for varying the attack interval of a single attacker in simulation (left-hand plot) and lab measurement (right-hand plot). The results are shown for two probabilities of sending Endpoint Unreachables for each selected entry: u=0% and u=100% (i.e. worst case).

Due to the “stateful” behaviour of the RAND and LUD policies, the attacker is able to reduce the handling speed until triggering the countermeasure mechanism. Then, the attacker is ignored and the performance remains as for attack-free scenarios. For the LU and RAND policies, even very small attack intervals (e.g. A=0.001s) have no significant impact in the simulation. Only for LU and u=100%, a performance degradation can be observed in the measurement scenario: this effect is caused by the latency of the PE load state updates in the real network (which can vary depending on the SCTP protocol’s flow control – which is not covered by the simulation model): since the load value of a PE only changes on reregistration, a least-loaded PE entry may be selected multiple times in sequence. That is, the attacker will send multiple unreachability reports for the same PE. As long as the threshold of $\mathrm{MaxEURate}$ is not yet reached, the PE entry may be removed. However, when the countermeasure threshold is reached, the attacker is ignored and the performance goes back to the level of attacker-free scenarios.

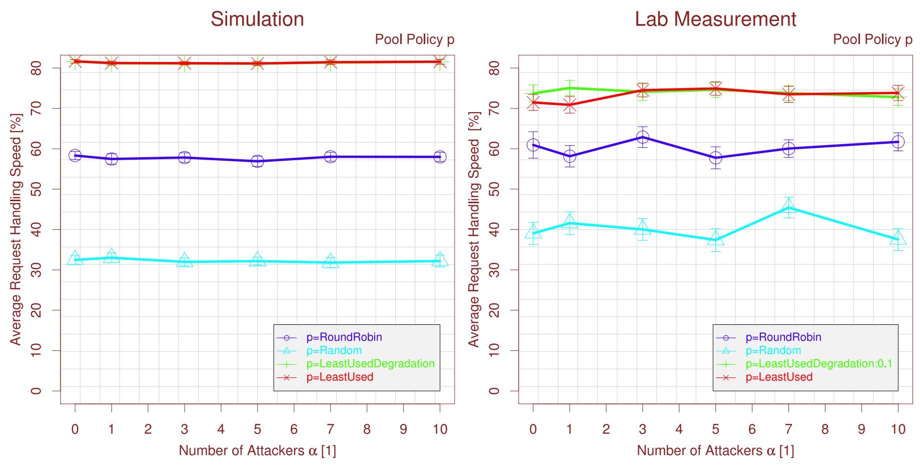

Figure 6. Applying Countermeasures for Varying the Number of Pool-User-Based Attackers

Figure 6 presents the results for varying the number of attackers α with an attack interval of A=0.1 (for which a *single* attacker was able to cause a complete DoS without countermeasures, as shown in subsection 5.2). The attacker sends Endpoint Unreachables at a probability of u=100% for the selected PE entries (worst case). Clearly, the presented results show that even α=10 attackers have no significant impact on the performance – neither in simulation nor in reality – any more, due to the applied countermeasures.

7 Conclusions

In this paper, we have examined the two critical attack threats on RSerPool systems by both, simulations and corresponding lab measurements:

- PE-based attacks (registration) and
- PU-based attacks (handle resolution/failure report).

Without any protection, even a single attacker is easily able to achieve a complete DoS. For both types of attacks, we have introduced countermeasure approaches which are efficiently realizable with small memory and CPU requirements – as it is necessary for the "lightweight" RSerPool architecture. By simulations and lab measurements we have shown that these mechanisms work as expected and provide a significant performance gain – in comparison to an unprotected setup – in case of DoS attacks.

The ideas of our RSerPool research have been contributed into IETF's RSerPool standardization process, which has just reached a major milestone by publication of its basic protocol documents as RFCs. As part of our future work, it is also necessary to analyse the robustness of the ENRP protocol. Although the threat on the small number of PRs of an operation scope is significantly smaller, it is useful to obtain knowledge of possible attack scenarios. Furthermore, we intend to perform real-world security experiments in the PLANETLAB – again by using our RSerPool implementation RSPLIB. Our goal is to provide security and configuration guidelines for application developers and users of the IETF's new RSerPool standard.

References

1. S. Bellovin, J. Ioannidi, A. Keromytis, and R. Stewart. On the Use of Stream Control Transmission Protocol (SCTP) with IPsec. Standards Track RFC 3554, IETF, July 2003.
2. S. A. Crosby and D. S. Wallach. Denial of service via Algorithmic Complexity Attacks. In *Proceedings of the 12th USENIX Security Symposium*, pages 29–44, Washington, DC/U.S.A., Aug. 2003.
3. T. Dreibholz. *Reliable Server Pooling – Evaluation, Optimization and Extension of a Novel IETF Architecture*. PhD thesis, University of Duisburg-Essen, Faculty of Economics, Institute for Computer Science and Business Information Systems, Mar. 2007.
4. T. Dreibholz and E. P. Rathgeb. On the Performance of Reliable Server Pooling Systems. In *Proceedings of the IEEE Conference on Local Computer Networks (LCN) 30th Anniversary*, pages 200–208, Sydney/Australia, Nov. 2005. ISBN 0-7695-2421-4.
5. T. Dreibholz and E. P. Rathgeb. On Improving the Performance of Reliable Server Pooling Systems for Distance-Sensitive Distributed Applications. In *Proceedings of the 15. ITG/GI Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, pages 39–50, Bern/Switzerland, Feb. 2007. ISBN 978-3-540-69962-0.
6. T. Dreibholz and E. P. Rathgeb. A Powerful Tool-Chain for Setup, Distributed Processing, Analysis and Debugging of OMNeT++ Simulations. In *Proceedings of the 1st ACM/ICST OMNeT++ Workshop*, Marseille/France, Mar. 2008. ISBN 978-963-9799-20-2.
7. T. Dreibholz and E. P. Rathgeb. An Evaluation of the Pool Maintenance Overhead in Reliable Server Pooling Systems. *SERSC International Journal on Hybrid Information Technology (IJHIT)*, 1(2):17–32, Apr. 2008.

8. T. Dreibholz and E. P. Rathgeb. Reliable Server Pooling – A Novel IETF Architecture for Availability-Sensitive Services. In *Proceedings of the 2nd IEEE International Conference on Digital Society (ICDS)*, pages 150–156, Sainte Luce/Martinique, Feb. 2008. ISBN 978-0-7695-3087-1.
9. T. Dreibholz, E. P. Rathgeb, and X. Zhou. On Robustness and Countermeasures of Reliable Server Pooling Systems against Denial of Service Attacks. In *Proceedings of the IFIP Networking*, pages 586–598, Singapore, May 2008. ISBN 978-3-540-79548-3.
10. T. Dreibholz and M. Tüxen. Reliable Server Pooling Policies. RFC 5356, IETF, Sept. 2008.
11. T. Dreibholz, X. Zhou, and E. P. Rathgeb. A Performance Evaluation of RSerPool Server Selection Policies in Varying Heterogeneous Capacity Scenarios. In *Proceedings of the 33rd IEEE EuroMirco Conference on Software Engineering and Advanced Applications*, pages 157–164, Lübeck/Germany, Aug. 2007. ISBN 0-7695-2977-1.
12. I. Foster. What is the Grid? A Three Point Checklist. *GRID Today*, July 2002.
13. C. Hohendorf, E. P. Rathgeb, E. Unurkhaan, and M. Tüxen. Secure End-to-End Transport Over SCTP. *Journal of Computers*, 2(4):31–40, June 2007.
14. A. Jungmaier, E. Rescorla, and M. Tüxen. Transport Layer Security over Stream Control Transmission Protocol. Standards Track RFC 3436, IETF, Dec. 2002.
15. P. Lei, L. Ong, M. Tüxen, and T. Dreibholz. An Overview of Reliable Server Pooling Protocols. Informational RFC 5351, IETF, Sept. 2008.
16. P. Schöttle, T. Dreibholz, and E. P. Rathgeb. On the Application of Anomaly Detection in Reliable Server Pooling Systems for Improved Robustness against Denial of Service Attacks. In *Proceedings of the 33rd IEEE Conference on Local Computer Networks (LCN)*, pages 207–214, Montreal/Canada, Oct. 2008. ISBN 978-1-4244-2413-9.
17. R. Stewart. Stream Control Transmission Protocol. Standards Track RFC 4960, IETF, Sept. 2007.
18. R. Stewart, Q. Xie, M. Stillman, and M. Tüxen. Aggregate Server Access Protcol (ASAP). RFC 5352, IETF, Sept. 2008.
19. M. Stillman, R. Gopal, E. Guttman, M. Holdrege, and S. Sengodan. Threats Introduced by RSerPool and Requirements for Security. RFC 5355, IETF, Sept. 2008.
20. E. Unurkhaan. *Secure End-to-End Transport - A new security extension for SCTP*. PhD thesis, University of Duisburg-Essen, Institute for Experimental Mathematics, July 2005.
21. Q. Xie, R. Stewart, M. Stillman, M. Tüxen, and A. Silverton. Endpoint Handlespace Redundancy Protocol (ENRP). RFC 5353, IETF, Sept. 2008.
22. X. Zhou, T. Dreibholz, and E. P. Rathgeb. A New Approach of Performance Improvement for Server Selection in Reliable Server Pooling Systems. In *Proceedings of the 15th IEEE International Conference on Advanced Computing and Communication (ADCOM)*, pages 117–121, Guwahati/India, Dec. 2007. ISBN 0-7695-3059-1.
23. X. Zhou, T. Dreibholz, and E. P. Rathgeb. Improving the Load Balancing Performance of Reliable Server Pooling in Heterogeneous Capacity Environments. In *Proceedings of the 3rd Asian Internet Engineering Conference (AINTEC)*, volume 4866 of *Lecture Notes in Computer Science*, pages 125–140. Springer, Nov. 2007. ISBN 978-3-540-76808-1.
24. X. Zhou, T. Dreibholz, and E. P. Rathgeb. A New Server Selection Strategy for Reliable Server Pooling in Widely Distributed Environments. In *Proceedings of the 2nd IEEE International Conference on Digital Society (ICDS)*, pages 171–177, Sainte Luce/Martinique, Feb. 2008. ISBN 978-0-7695-3087-1.