A PlanetLab-Based Performance Analysis of RSerPool Security Mechanisms

Thomas Dreibholz*, Xing Zhou†, Erwin P. Rathgeb*, Wencai Du†
*University of Duisburg-Essen, Institute for Experimental Mathematics
Ellernstrasse 29, 45326 Essen, Germany
†Hainan University, College of Information Science and Technology
Renmin Avenue 58, 570228 Haikou, Hainan, China
dreibh@iem.uni-due.de, zhouxing@hainu.edu.cn, rathgeb@iem.uni-due.de, wencai@hainu.edu.cn

***Abstract*— Reliable Server Pooling (RSerPool) denotes the new IETF standard for a lightweight server redundancy and session failover framework for availability-critical applications. A number of research papers have already addressed the service and pool management performance of RSerPool in general. However, the important topic of security, including the system robustness against intentional attacks, has not yet been intensively addressed. In particular, none of the proposed Denial of Service (DoS) attack countermeasure mechanisms has been evaluated in a real-world Internet setup.**

For that reason, this paper provides an analysis of the robustness of RSerPool systems against DoS attacks. We will outline the DoS attack bandwidth which is necessary for a significant service degradation. Furthermore, we will present simple but effective DoS attack countermeasure mechanisms to significantly reduce the impact of attacks. Our analysis is based on a real-world Internet setup using the PlanetLab. We will furthermore compare the performance measurements against simulation results.[1]

Keywords: Reliable Server Pooling, Security, Attacks, Denial of Service, Robustness, Performance Analysis

I. Introduction and Scope

In the Internet of today, there is a growing demand for highly available services. To cope with the requirements of availability-critical services, the IETF has just published a generic, application-independent server pool [1] and session management [2] framework as RFCs: Reliable Server Pooling (RSerPool, see [3]). It is responsible for the required server redundancy and session management. While there have already been a number of publications on the performance of RSerPool for load balancing [2], [4] and server failure handling [5], there has been very little research on its security and Denial of Service (DoS) attack robustness. Until now, only basic concepts to avoid flooding attacks on the pool management have been analysed by simulations in [6], [7] and a lab setup in [8]. However, the mechanisms have not been tested in a real-world distributed Internet setup – which is quite realistic for availability-critical RSerPool-based services [9].

The underlying transport protocol SCTP[2] already contains countermeasures against blind flooding attacks [11] and the RFC [12] of RSerPool mandatorily requires applying mechanisms like TLS [13] or IPsec [14] in order to ensure authenticity, integrity and confidentiality. Nevertheless, relying on these techniques alone is not sufficient: there is still a chance that an attacker may compromise a legitimate component, e.g. by exploiting software bugs to steal its private key. Therefore, it is important to analyse the implications to the service under DoS attack situations, in order to apply effective attack countermeasures.

The goal of this paper is the DoS attack robustness analysis of the RSerPool architecture by simulations as well as their experimental validation in real-life by using the PlanetLab [15]. First, we will observe the impact of different attack scenarios on the application performance. Using these analyses as the baseline performance level, we will present effective techniques to reduce the impact of such attacks.

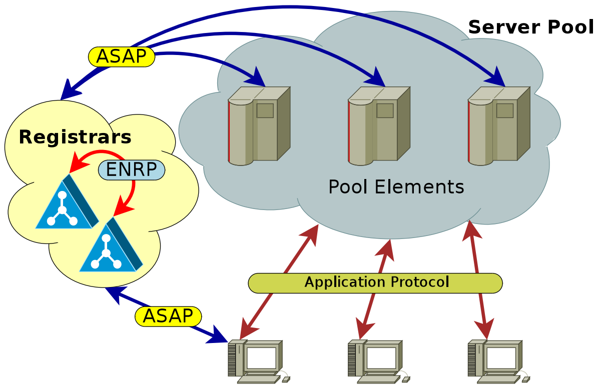

Figure 1. The RSerPool Architecture

II. The RSerPool Architecture

An overview of the RSerPool architecture [2], [3] with its three types of components is depicted in figure 1: a server in a pool is called *pool element* (PE), a client is denoted as a *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).

A. Registrar Operations

PRs of an *operation scope* synchronize their view of the handlespace by using the Endpoint haNdlespace Redun-

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

[2]Stream Control Transmission Protocol, see [10].

dancy Protocol (ENRP) [16], transported via SCTP [10] and secured e.g. by TLS [13] or IPsec [14]. In contrast to Grid Computing [17], 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). This property leads to small management overhead [1], [18], which also allows for RSerPool usage on devices which have limited memory and CPU resources (e.g. telecommunications equipment). Nevertheless, PEs may be distributed globally to continue their service even in case of localized disasters [9] (e.g. an earthquake).

B. Pool Element Operations

PEs choose an arbitrary PR of the operation scope to register into a pool by using the Aggregate Server Access Protocol (ASAP) [19], 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.

C. Pool User Operations

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*. Two classes of load distribution policies are supported: non-adaptive and adaptive strategies [1], [4]. While adaptive strategies base their selections on the current PE state (which requires up-to-date information), non-adaptive algorithms do not need such data. A basic set of adaptive and non-adaptive pool policies is defined in [20]. 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 [21] 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.

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 $\mathrm{MaxBadPEReports}$ [5] (default is 3 [19]), the PR may decide to remove the PE from the handlespace. The counter of a PE is reset upon its re-registration.

D. A Handlespace Example

An example handlespace containing four pools is illustrated in figure 2. The pool using the PH "Compute Pool" consists of 3 dual-homed PEs (IPv4 and IPv6). Since its pool policy is LU, the handlespace also stores the latest known load state of each PE.

E. Application Scenarios

While the initial motivation of RSerPool has been the availability of SS7 (Signalling System No. 7 [22]) services over IP networks, it has been designed for application independence. Current research on applicability and performance of RSerPool includes application scenarios (described in detail by [2, section 3.6]) like VoIP with SIP, SCTP-based mobility, web server pools, e-commerce systems [23], video on demand [24], battlefield networks [25], IP Flow Information Export (IPFIX) and workload distribution [2], [4], [26], [27].

III. Quantifying an RSerPool System

As application model for our quantitative performance analysis, we use the model from [2]: 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*. Requests are queued and sequentially assigned.

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 $d_{\mathrm{Processing}}$ (acceptance until finish):

$$d_{\mathrm{Handling}} = d_{\mathrm{Queuing}} + d_{\mathrm{Startup}} + d_{\mathrm{Processing}}. \quad (1)$$

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:

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

For convenience, 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 NumPEs servers and total pool capacity $\mathrm{PoolCapacity}$) as:

$$\mathrm{SysUtil} = \mathrm{NumPEs} * \mathrm{puToPERatio} * \frac{\frac{\mathrm{RequestSize}}{\mathrm{RequestInterval}}}{\mathrm{PoolCapacity}}. \quad (2)$$

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

[3]An application-specific view of capacity may be mapped to this definition, e.g. CPU cycles.

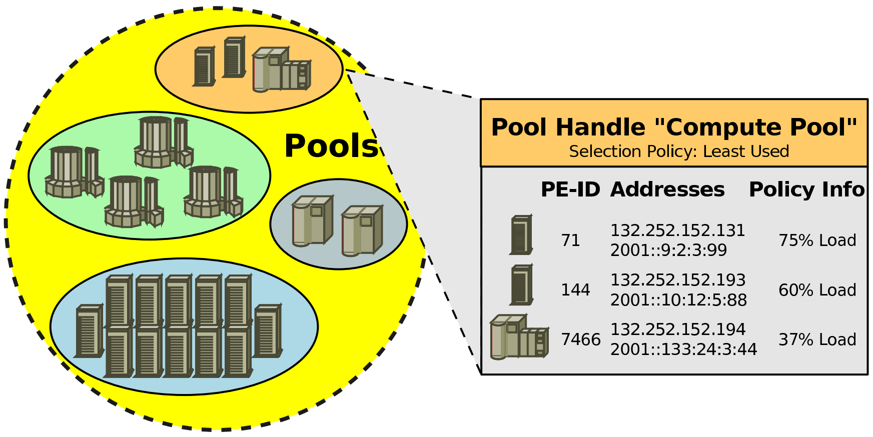

Figure 2. A Handlespace Example

dimensioned for a certain *target system utilization* of e.g. 50%. 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 also [2]).

IV. System Setup

For our performance analysis, we have used our OMNeT++-based RSerPool simulation model RSPSIM [4], [27] as well as our implementation RSPLIB [2], [9], [28] (which is also the IETF's reference implementation, see [3, chapter 5]) for measurements in a PlanetLab setup. Both – simulation model and implementation – contain the protocols ASAP [19] and ENRP [16], a PR module, an attacker module and PE as well as PU modules for the request handling scenario defined in section III.

The PlanetLab [15] setup distributes the components to different machines in the U.S.A.. This country provides a sufficient number of PlanetLab nodes and a country-wide setup is also realistic for an RSerPool setup in a large company, in order to protect a critical service against e.g. earthquakes, power failures or terrorist attacks. By `ping`-based tests, we have observed inter-node network delays of about 20ms to 30ms.

The Linux-based PlanetLab nodes only support the protocols TCP and UDP, i.e. in particular the Linux Kernel SCTP module (LK-SCTP, see [29]) is not provided. Unlike for our lab measurements in [8], we therefore had to use our own userland SCTP implementation SCTPLIB [30], [31]. The restriction of PlanetLab to TCP and UDP furthermore made it necessary to tunnel our SCTP traffic over UDP using the "SCTP over UDP" encapsulation defined in [32]. Since our ASAP and ENRP messages are small compared to the usual MTU (i.e. 1500 bytes) and bandwidth is not the limiting factor, the additional per-packet overhead of 8 bytes is negligible.

For our simulation and measurement setup, which is depicted in figure 3, we use the following parameter settings unless otherwise specified:

- 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 [2], [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 a 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 such scenarios). PEs re-register every 30s (registration lifetime) and on every load change of the adaptive LU and LUD policies.
- MaxBadPEReports is set to 3 (default in RFC [19]). A PU sends an Endpoint Unreachable if a contacted PE fails to respond within 10s (see also [5]).
- 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. The inter-component network delay is 25ms, which corresponds to the PlanetLab observations above.
- Each measurement run takes 15min; each run is repeated at least 12 times.

For statistical post-processing of the results, GNU R [27], [33] is used. Each resulting plot shows the average values and their 95% confidence intervals.

V. The Performance Impact of Attacks

Targets of attacks on an RSerPool system are the PRs, PEs and PUs. Since the scope of RSerPool is restricted to a single administrative domain, sufficient protection of the small number of PRs is assumed to be feasible with a reasonable effort (see also [6], [7]). The PEs and PUs are *obviously* more likely attack targets: their number may become large (e.g. for real-time distributed computing, see [2], [26]) and they may be distributed over a wide geographical and less controllable area (e.g. to survive localized disasters, see [9]). Therefore, ASAP-based attacks are the focus of our study in this paper. As we will show, even a *single* compromised PE or PU can achieve a DoS of the whole system if no protection mechanisms are applied.

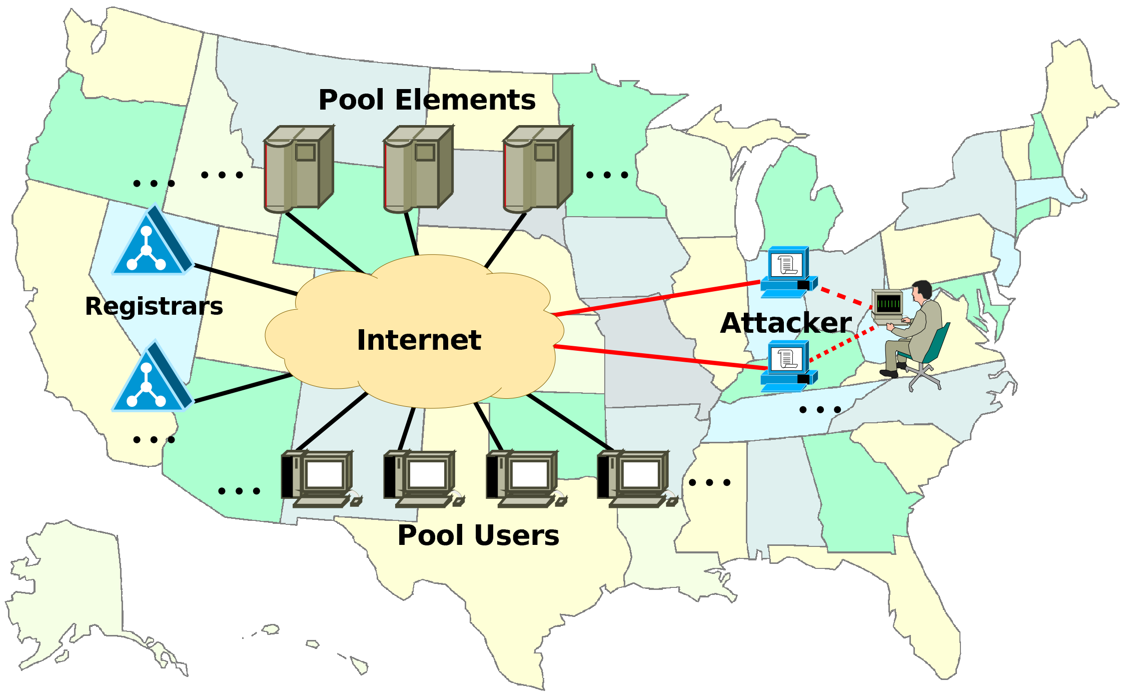

Figure 3. The RSerPool System Setup used for the Analyses

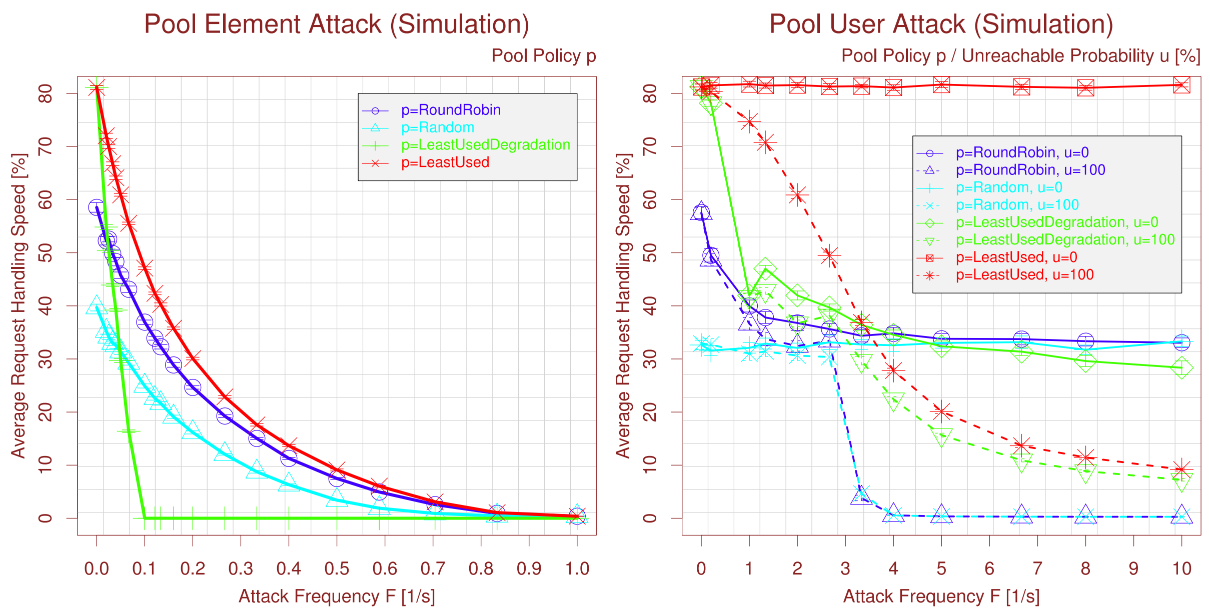

Figure 4. The Impact of a PE/PU-Based Attacks without Countermeasures

A. An Attack by a Compromised Pool Element

Clearly, an attacker masquerading as a PE will try to perform as many fake registrations as possible: each registration request only has to contain another (e.g. randomly chosen) PE ID. The policy parameters can be set appropriately, i.e. a load of 0% (LU, LUD) and a load increment of 0% (LUD), in order to ensure that the fake PE entry is chosen (upon a handle resolution) as frequently as possible. SCTP [10], [11] as the underlying protocol already prevents simple address spoofing attacks: each network-layer address under which a PE is registered must be part of the SCTP association between PE and PR. The ASAP protocol [19] requires the addresses to be validated by SCTP. However, holding a registration association to the PR and silently dropping all incoming PU requests is sufficient for an attacker.

The impact of varying the attack frequency F (i.e. the number of fake registrations per second) on the handling speed in the simulation scenario is shown on the left-hand side of figure 4. Obviously, a frequency of F=0.1 (i.e. one registration every 10s) is already sufficient to significantly decrease the service performance of the setup. Using the LUD policy [21], this already leads to a complete DoS: an unloaded PE (load is 0%) whose load does not increase when accepting a new request (load increment is 0%) appears to the PU as a really good choice. As result, the PUs will exclusively select the fake PE entries. Using a higher setting of F (e.g. here: 0.8), a DoS is also reached for the other policies, too. The corresponding PLANETLAB measurement results show a similar behaviour. A plot has therefore been omitted.

B. An Attack by a Compromised Pool User

Obviously, the attack threat by a compromised PU would be to flood the PR with handle resolution requests. But the server selection functionality as part of the handlespace management can be realized very efficiently [1], [18]; an attacker would therefore require a high attack bandwidth to effectively degrade the service performance. However, even by just requesting *a few* handle resolutions – without actually using any selected PE's service – the performance of the service may be affected: as example, the impact of a handle resolution attack on the performance for varying the attack frequency F (i.e. the delay between two handle resolution requests) is shown on the right-hand side of figure 4. For the PE entries chosen by a handle resolution, an unreachability report (see section II) is sent with probability u. The two extreme cases are most interesting: u=0% (no unreachability reports – represented by solid lines) and u=100% (worst case – represented by dotted lines).

The performance of RR is already affected by a setting of u=0%, due to the "stateful" [4] operation of RR: some PEs of the "in turn" selection are skipped (since they are not actually used for providing their service), leading to the usage of less appropriate PEs for real requests. LUD is affected in a similar way by increased load values. Since LU and RAND are "stateless", they are not affected by this kind of attack. But the service performance impact of also reporting all PEs as being unreachable – i.e. u=100% – is dramatic: PEs are kicked out of the handlespace, and the handling speed quickly sinks and leads – here at about F=10, i.e. only 10 reports/s – to a complete DoS. A similar behaviour can also be observed for the corresponding PLANETLAB measurements; their results plots have been omitted therefore.

VI. Our Attack Countermeasure Mechanisms

As shown above, even only a single attacker with a small attack bandwidth (i.e. a few messages/s, which is easily feasible over a low-bandwidth modem connection) can lead to a complete DoS. That is, effective countermeasures are necessary to avoid such situations.

A. Pool Element Attack Countermeasures

The threat of a PE-based attack – as explained in subsection V-A – is that the attacker can easily flood the handlespace with fake PE entries. The small bandwidth of a modem connection is already sufficient to cause at least a significant service degradation. Starting point for a countermeasure to this attack type is therefore a restriction of the number of PE registrations a single PE identity is allowed to create. In order to retain the "lightweight" [1], [18] property of RSerPool and to avoid synchronizing such numbers among PRs, our countermeasure approach first introduces a so-called *registration authorization ticket* (suggested by us in [6]), which consists of:

1) the pool's PH and a fixed PE ID,
2) minimum/maximum policy information settings (e.g. lower bound on LUD load decrement) and
3) a signature of the ticket by a trustworthy authority (explained below).

The registration authorization ticket is provided by a PE to its PR-H as part of the ASAP registration request. Since ASAP messages use a TLV structure (see [2, section 3.8] for a detailed description), the effort of adding an additional field for the ticket is minimal. The validity of the ticket can be verified easily by checking its signature. 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 has to verify the authorization ticket (with additional time complexity in $O(1)$). In particular, it is neither necessary to change the protocols (except for adding the ticket field described above) nor to perform additional ENRP-based synchronization of authorization information among the PRs of the operation scope.

The only infrastructure requirement which is added by our approach is the need for a trusted authority, which can e.g. be realized by a Kerberos [34] service. This is feasible at reasonable effort, since an operation scope is restricted to a single administrative domain (as explained in section II).

In order to show that our approach is effective, figure 5 presents the handling speed results of a PLANETLAB measurement for an attack frequency of F=10 per attacker for varying the number of attackers α from 0 to 10. Note that α=10 attacker PEs means to have as many attackers as there are legitimate PEs in the pool. In particular, the attacker would have to compromise 10 real PEs to steal their registration authorization tickets in order to perform such an attack. That is, a significant effort by the attacker is necessary. As shown in subsection V-A, F=10 has already lead to a full DoS with only a *single* attacker.

As it is clearly observable, our countermeasure approach is quite effective: even for $\alpha = 10$, the handling speed only halves at most – but the service which is provided by the 10 real PEs in the pool still remains operational and the attack impact is not even close to a DoS. The results obtained from the PLANETLAB measurements correspond to the simulation results, i.e. the mechanism also works effectively in a real-world Internet setup.

Note, that the slightly different handling speed levels of the RSPLIB-based PLANETLAB measurements in comparison to the RSPSIM simulation results are caused by node (e.g. background PLANETLAB slices, kernel, operating system, applications) and SCTP association (e.g. packet scheduling, network delay and jitter, retransmissions) latencies. These are – due to their complexity – not fully incorporated into the RSPSIM simulation model. Nevertheless, the tendency of the results is observable reasonably well.

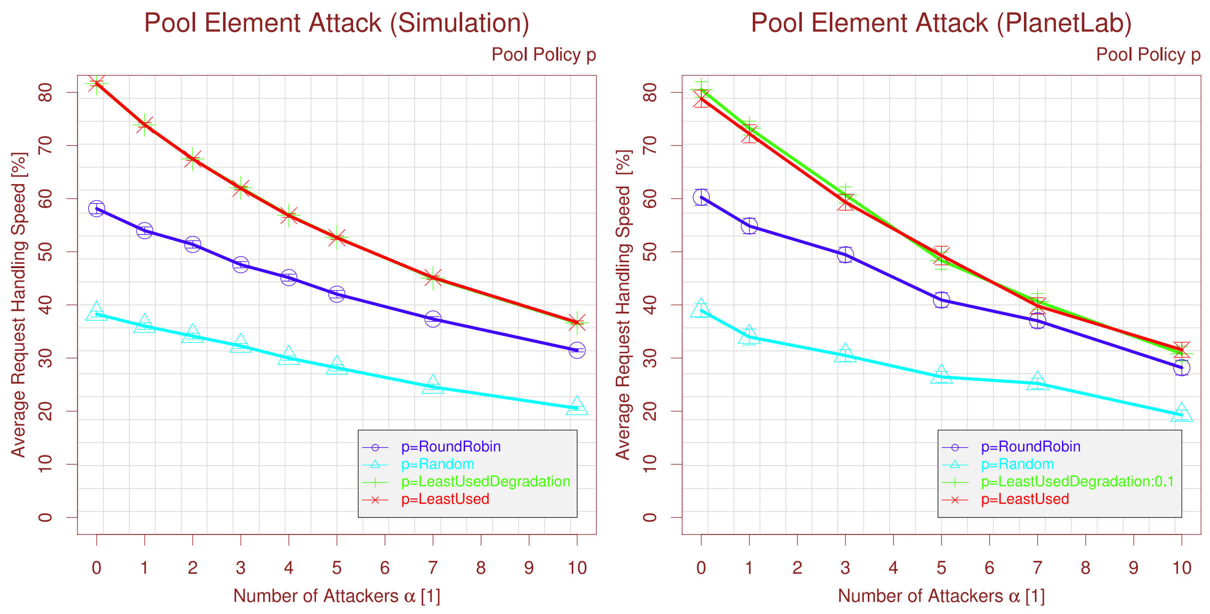

Figure 5. Applying Countermeasures Against Pool-Element-Based Attacks

B. *Pool User Attack Countermeasures*

By using failure reports, malicious PUs have the ability to impeach PEs (as shown in subsection V-B). Our strategy for a countermeasure is therefore to first introduce a PU identification which is certified by a trusted authority and can be verified by the PR (analogously to the PE's registration authorization ticket explained in subsection VI-A). Then, the number of failure reports sent by a PU can be tracked in order to avoid counting multiple reports from the same PU. To realize this tracking functionality, a PR simply has to memorize (explained later) the PH for which a certain PU has reported unreachable PEs. After that, multiple reports coming for the same pool can simply be ignored. Since the unreachability count for each PE is a PR-local variable, no synchronization among PRs is necessary. That is, an attacker is unable to cause harm to the service by simply sending its unreachability reports for the same PE to different PRs.

To avoid the necessity to store each reported PE identity – which would be exploitable in form of a so-called *computational complexity attack* [35] by sending a large number of random PE IDs – we use a hash-based per-PU message blackboard (as suggested by us in [8]): the function Ψ maps a PE's PH into a bucket:

$$\Psi(\mathrm{PH}) = h(\mathrm{PH}) \text{ MOD } \mathrm{NumberOfBuckets}.$$

h denotes a hash function that is not easily guessable by the attacker. This property is provided by so-called *universal hash functions* [35], which are – unlike cryptographic hash functions (e.g. SHA-1 [36]) 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:

$$\mathrm{Rate} = \frac{\mathrm{NumberOfTimeStamps}}{\mathrm{TimeStamp}_{\mathrm{Last}} - \mathrm{TimeStamp}_{\mathrm{First}}}. \qquad (3)$$

When a PR receives an Endpoint Unreachable from a PU, it simply updates the reported PE's corresponding bucket entry. If the rate in equation 3 exceeds the configured threshold $\mathrm{MaxEURate}$, the report is silently ignored. The time complexity for this operation is in $O(1)$, as well as the required per-PU storage space. Analogously, the same hash-based approach can be applied for handle resolutions with the corresponding threshold $\mathrm{MaxHRRate}$. But instead of simply ignoring the request, the PR here replies with an empty list. This indicates a currently empty pool.

An alternative approach is presented in [7]: instead of specifying a fixed threshold, statistical anomaly detection is applied: the behaviour of the majority of nodes is assumed to be "normal". Differing behaviour – which is necessary for an effective attack – is denoted as an 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. But the advantage of this approach is that the system can automatically adapt to a changing environment, e.g. when new applications are deployed.

Figure 6 presents the performance 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) for varying the attack frequency of *one* attacker in a simulation (left-hand plot) and a PLANETLAB measurement (right-hand plot). The probabilities for sending Endpoint Unreachables are u=100% (worst case) and u=0% (for comparison).

Since the policies RR and LUD are "stateful", the attacker is able to reduce the handling speed until triggering the countermeasure mechanism. After that, the attacker is ignored and the performance remains as for attacker-free scenarios. For the "stateless" RAND policy, the attack has no impact; for the LU policy, the attacker only has an impact when using unreachability reports. Interestingly,

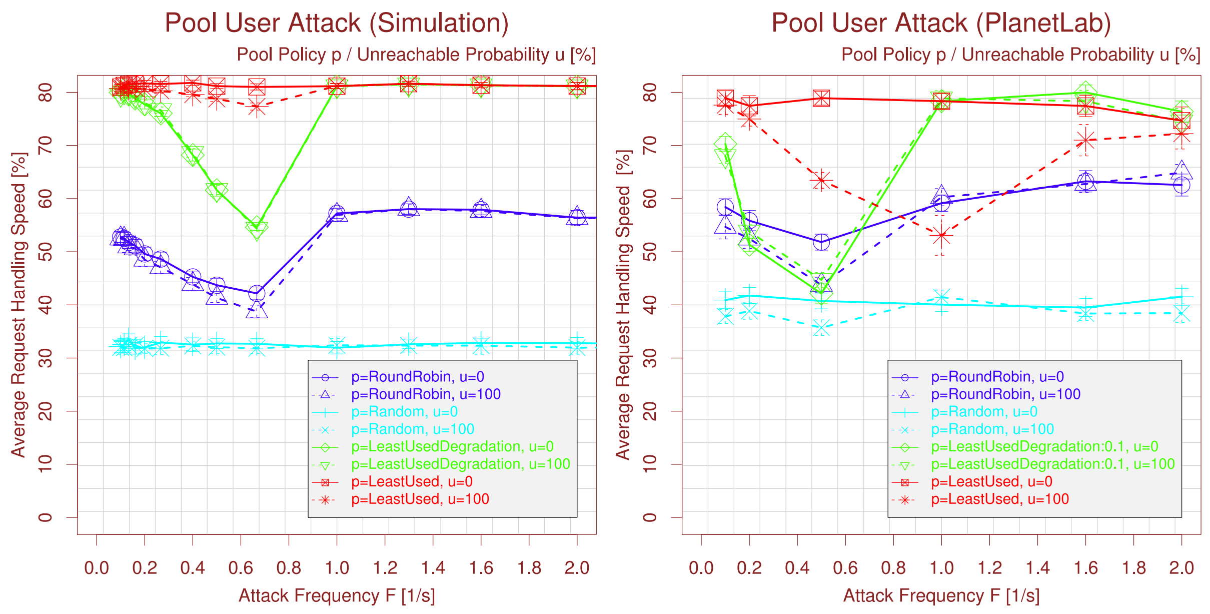

Figure 6. Applying Countermeasures Against a Pool-User-Based Attack

this effect is stronger for the PLANETLAB measurements than for the simulation: this effect is caused by the latency of the PE load state updates in the Internet (which may vary due to node latencies and congestion): 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 rate threshold is not yet reached, the PE entry may be impeached. However, when the countermeasure threshold is reached, the attacker is ignored and the performance returns to the level of an attacker-free system setup.

The results for varying the number of attackers α with an attack frequency of F=10 (for which even only *one* attacker has been able to cause a complete DoS without countermeasures, as shown in subsection V-B) are depicted in figure 7. The attacker sends Endpoint Unreachables at a probability of u=100% for the selected PE entries (i.e. the worst case). Clearly, the presented results show that even α=10 attackers have no significant impact on the performance – neither in simulation nor in the real-world Internet setup – any more, due to the countermeasures. Again, 10 attackers would mean that 10 legitimate components had been compromised and their authorization tickets had been stolen – which is a high and non-trivial effort for an attacker in the restricted operation scope of an RSerPool setup (see section II).

VII. CONCLUSIONS

The two most important DoS attack threats for RSerPool systems are PE-based attacks (registration) and PU-based attacks (handle resolution/failure report). We have examined these threats using simulations as well as in real-life by PLANETLAB measurements. Without further protection mechanisms, even only a single attacker with a small attack bandwidth can easily cause a DoS. For both types of attacks, we have introduced countermeasure approaches which have shown to be effective – in simulations as well as in reality. Furthermore, they are efficiently realizable – which is necessary to achieve the "lightweight" property of the RSerPool architecture.

The IETF's standardization process for RSerPool has just reached a major milestone by publication of its basic protocol documents as RFCs. Since the early beginnings of this process we have contributed our ideas, evaluations and improvements for the RSerPool framework. In the future RSerPool research, it is also necessary to further 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 a broad knowledge of possible DoS attack vectors to deploy reasonable and effective countermeasures. The goal of our ongoing work is to provide comprehensive security and configuration guidelines for application developers and users of the IETF's new RSerPool standard.

REFERENCES

[1] 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)*, vol. 1, no. 2, pp. 17–32, Apr. 2008.

[2] T. Dreibholz, "Reliable Server Pooling – Evaluation, Optimization and Extension of a Novel IETF Architecture," Ph.D. dissertation, University of Duisburg-Essen, Faculty of Economics, Institute for Computer Science and Business Information Systems, Mar. 2007.

[3] P. Lei, L. Ong, M. Tüxen, and T. Dreibholz, "An Overview of Reliable Server Pooling Protocols," IETF, Informational RFC 5351, Sept. 2008.

[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*, Sydney/Australia, Nov. 2005, pp. 200–208, ISBN 0-7695-2421-4.

[5] ——, "Reliable Server Pooling – A Novel IETF Architecture for Availability-Sensitive Services," in *Proceedings of the 2nd IEEE International Conference on Digital Society (ICDS)*, Sainte Luce/Martinique, Feb. 2008, pp. 150–156, ISBN 978-0-7695-3087-1.

[6] 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*, Singapore, May 2008, pp. 586–598, ISBN 978-3-540-79548-3.

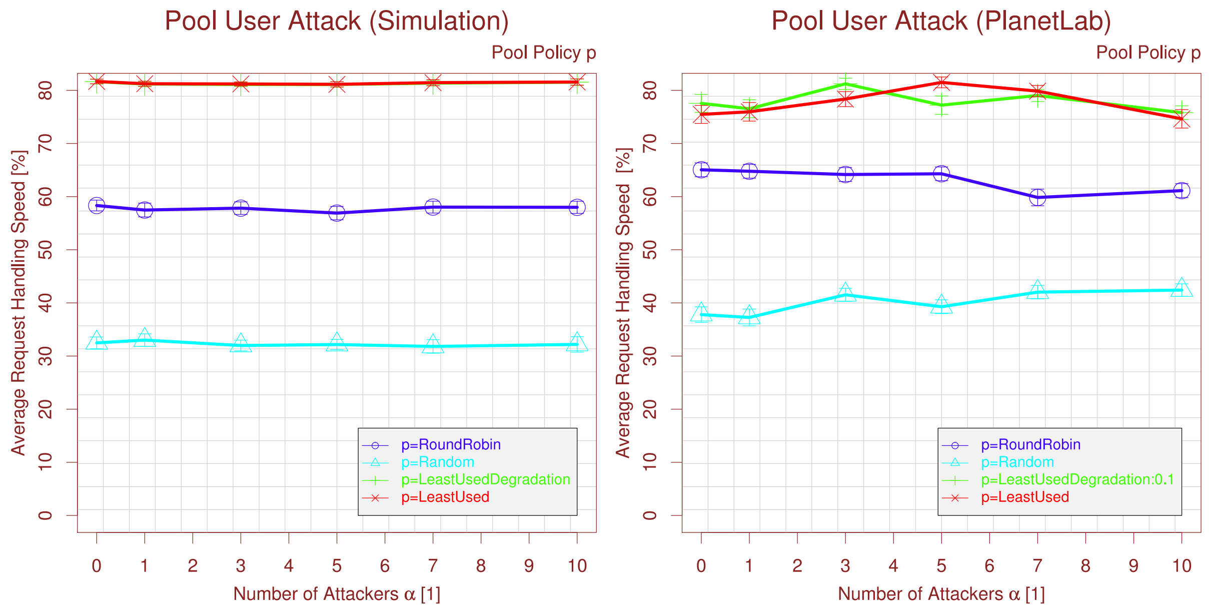

Figure 7. Varying the Number of Pool-User-Based Attackers

[7] 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)*, Montreal/Canada, Oct. 2008, pp. 207–214, ISBN 978-1-4244-2413-9.

[8] X. Zhou, T. Dreibholz, W. Du, and E. P. Rathgeb, "Evaluation of Attack Countermeasures to Improve the DoS Robustness of RSerPool Systems by Simulations and Measurements," in *Proceedings of the 16. ITG/GI Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, Kassel/Germany, Mar. 2009, pp. 217–228, ISBN 978-3-540-92665-8.

[9] 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)*, Bern/Switzerland, Feb. 2007, pp. 39–50, ISBN 978-3-540-69962-0.

[10] R. Stewart, "Stream Control Transmission Protocol," IETF, Standards Track RFC 4960, Sept. 2007.

[11] E. Unurkhaan, "Secure End-to-End Transport - A new security extension for SCTP," Ph.D. dissertation, University of Duisburg-Essen, Institute for Experimental Mathematics, July 2005.

[12] M. Stillman, R. Gopal, E. Guttman, M. Holdrege, and S. Sengodan, "Threats Introduced by RSerPool and Requirements for Security," IETF, RFC 5355, Sept. 2008.

[13] A. Jungmaier, E. Rescorla, and M. Tüxen, "Transport Layer Security over Stream Control Transmission Protocol," IETF, Standards Track RFC 3436, Dec. 2002.

[14] S. Bellovin, J. Ioannidi, A. Keromytis, and R. Stewart, "On the Use of Stream Control Transmission Protocol (SCTP) with IPsec," IETF, Standards Track RFC 3554, July 2003.

[15] L. Peterson and T. Roscoe, "The Design Principles of PlanetLab," *Operating Systems Review*, vol. 40, no. 1, pp. 11–16, Jan. 2006.

[16] Q. Xie, R. Stewart, M. Stillman, M. Tüxen, and A. Silverton, "Endpoint Handlespace Redundancy Protocol (ENRP)," IETF, RFC 5353, Sept. 2008.

[17] I. Foster, "What is the Grid? A Three Point Checklist," *GRID Today*, July 2002.

[18] T. Dreibholz and E. P. Rathgeb, "Implementing the Reliable Server Pooling Framework," in *Proceedings of the 8th IEEE International Conference on Telecommunications (ConTEL)*, vol. 1, Zagreb/Croatia, June 2005, pp. 21–28, ISBN 953-184-081-4.

[19] R. Stewart, Q. Xie, M. Stillman, and M. Tüxen, "Aggregate Server Access Protcol (ASAP)," IETF, RFC 5352, Sept. 2008.

[20] T. Dreibholz and M. Tüxen, "Reliable Server Pooling Policies," IETF, RFC 5356, Sept. 2008.

[21] 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)*, Sainte Luce/Martinique, Feb. 2008, pp. 171–177, ISBN 978-0-7695-3087-1.

[22] ITU-T, "Introduction to CCITT Signalling System No. 7," International Telecommunication Union, Tech. Rep. Recommendation Q.700, Mar. 1993.

[23] T. Dreibholz, "An Efficient Approach for State Sharing in Server Pools," in *Proceedings of the 27th IEEE Local Computer Networks Conference (LCN)*, Tampa, Florida/U.S.A., Oct. 2002, pp. 348–352, ISBN 0-7695-1591-6.

[24] A. Maharana and G. N. Rathna, "Fault-tolerant Video on Demand in RSerPool Architecture," in *Proceedings of the International Conference on Advanced Computing and Communications (ADCOM)*, Bangalore/India, Dec. 2006, pp. 534–539, ISBN 1-4244-0716-8.

[25] Ü. Uyar, J. Zheng, M. A. Fecko, S. Samtani, and P. Conrad, "Evaluation of Architectures for Reliable Server Pooling in Wired and Wireless Environments," *IEEE JSAC Special Issue on Recent Advances in Service Overlay Networks*, vol. 22, no. 1, pp. 164–175, 2004.

[26] T. Dreibholz, "Applicability of Reliable Server Pooling for Real-Time Distributed Computing," IETF, Individual Submission, Internet-Draft Version 06, Jan. 2009, draft-dreibholz-rserpool-applic-distcomp-06.txt, work in progress.

[27] 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.

[28] T. Dreibholz, "Thomas Dreibholz's RSerPool Page," 2009.

[29] LKSCTP, "Linux Kernel SCTP," 2009.

[30] A. Jungmaier, "Das Transportprotokoll SCTP," Ph.D. dissertation, Universität Duisburg-Essen, Institut für Experimentelle Mathematik, Aug. 2005.

[31] M. Tüxen, "The sctplib Prototype," 2009.

[32] M. Tüxen and R. Stewart, "UDP Encapsulation of SCTP Packets," IETF, Individual Submission, Internet-Draft Version 02, Nov. 2007, draft-tuexen-sctp-udp-encaps-02.txt, work in progress.

[33] T. Dreibholz, X. Zhou, and E. P. Rathgeb, "SimProcTC – The Design and Realization of a Powerful Tool-Chain for OMNeT++ Simulations," in *Proceedings of the 2nd ACM/ICST OMNeT++ Workshop*, Rome/Italy, Mar. 2009, ISBN 978-963-9799-45-5.

[34] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The Kerberos Network Authentication Service (V5)," IETF, Standards Track RFC 4120, July 2005.

[35] S. A. Crosby and D. S. Wallach, "Denial of service via Algorithmic Complexity Attacks," in *Proceedings of the 12th USENIX Security Symposium*, Washington, DC/U.S.A., Aug. 2003, pp. 29–44.

[36] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," IETF, Informational RFC 3174, Sept. 2001.