

Splash: SNMP Plus a Lightweight API for SNAP Handling

Willem de Bruijn
Universiteit Leiden
Leiden, The Netherlands
wdebruij@liacs.nl

Herbert Bos[†]
Vrije Universiteit
Amsterdam, The Netherlands
H.Bos@few.vu.nl
([†] formerly affiliated with Universiteit Leiden)

Jonathan T. Moore
University of Pennsylvania
Philadelphia, Pennsylvania
jonm@cis.upenn.edu

Abstract

We describe a drop-in replacement for `net-snmp`, known as ‘Splash’ (SNMP Plus a Lightweight API for SNAP Handling), which adds support for efficient mobile agents to the standard SNMP agent. Experiments show that the SNAP (Safe and Nimble Active Packets) mobile agent engine is comparable in speed to SNMP, even for the simple polling scenarios for which SNMP is optimized, yet also provides flexibility that allows it to outperform SNMP in more complex scenarios. Splash allows network operators to select the monitoring paradigm (polling with SNMP, or a mobile agent approach) most appropriate for a given situation.

Keywords

Programmable, Active and Adaptive Management, Active Networks, Mobile Agents, Network Management, SNMP

1. Introduction

1.1 Background

Efficient network management is vital to the performance of the network. High rates and stretched resources provide two important requirements for network management solutions: (1) the response to network problems should be fast, and (2) the amount of traffic generated by the system itself should be kept to a minimum.

Most of today’s networks rely on SNMP (Simple Network Management Protocol) for their management needs. SNMP provides simple GET and SET access to management variables on remote systems. For many scenarios this functionality suffices. However, in SNMP all the management *logic* is located at a central management station (MS), while all the management *data* is located at the various managed objects. For some applications this leads to scalability problems, as large amounts of data may need to be transmitted to the MS. For example, SNMP has sometimes been criticized for its excessive usage of network resources. For the same reason, it can

be slow in responding to problems. Another problem is that SNMP stipulates a rigid, hierarchically structured, management topology.

Mobile agents have often been proposed as a solution to some of these scalability issues (e.g. in [1, 3, 5, 11, 22, 12, 13, 15, 17, 24]). The main advantages touted by their advocates are: remote data aggregation, distributed decision making, and topology independence.

Skepticism toward mobile agent based solutions in general has led to poor acceptance of the technology in management. While some of the criticism is well founded and should be addressed, e.g. efficiency and safety of execution, it is not easy to overturn a subjective bias against the mobile agent paradigm. Unfortunately, mobile agent platforms tend to be heavyweight, proving beneficial only for very large networks [2, 14]. This, plus the fact that agents can be notoriously hard to control [25], has only added to the skepticism and prevented the widespread deployment of mobile agent-based management.

1.2 Contribution

The goal of this paper is not to make a case for mobile agent-based solutions in general, but to show that an efficient and lightweight mobile agent platform can play a useful role in the present network management toolbox. Management by delegation can take many forms, of which the predominant ones are examined by Simes *et al.* in [21]. They show that not all remote agent implementations are well suited to all occasions. However, in cases where stationary agents, for instance ScriptMIB instances, incur too much overhead, roaming agents can be put to use *if they can be shown to be sufficiently lightweight*.

It is not our intention to repeat the discussion of mobile agent applicability. Instead, our contribution is to demonstrate that at least some of the oft-cited arguments against mobile agent deployment, namely their (lack of) efficiency, safety and interoperability, are not universally valid. In this paper, we present a practical network management tool known as *Splash* (SNMP Plus a Lightweight API for SNAP Handling), which merges SNMP with *SNAP* (Safe and Nimble Active Packets) [9]. Compared to existing approaches, *Splash* is unique in having three advantages that may help it succeed in an area where other projects have failed to become widely adopted.

First, *Splash* can serve as a **drop-in replacement** for a standard SNMP daemon, supporting all existing SNMP-based applications, including ScriptMIB, AgentX and other extensions, at no cost in performance. Second, *Splash* active packets are very **lightweight** and efficient. They can be used to execute low level SNMP requests in the same order of time as SNMP. Third, *Splash* provides **hard resource bounds** as a way to prevent active packets from running out-of-control. The *Splash* active packets can be regarded as fairly dumb and very lightweight mobile agents*.

In Section 2 we describe the *Splash* architecture and its implementation. The application is evaluated in Section 3. Related work and conclusions are in Sections 4 and 5, respectively.

**Splash* can be downloaded from <http://splash-snap.sourceforge.net/>

2. Architecture and Implementation

2.1 Overview

Our architecture decomposes an SNMP agent in two parts: a protocol engine concerned with communication, and an MIB instrumentation that contains a collection of managed objects. In this architecture, supplemental protocol support is fairly easily incorporated by adding a new protocol engine. For mobile agents this engine comes in the form of a virtual machine, or VM (as shown in Figure 1). Unlike other approaches (e.g. [6]), the resulting system resides within a single address space, so that inter-process communication overheads are avoided.

2.2 SNAP

As the mobile agent platform for Splash we selected SNAP, a flexible programming language with high performance, yet safe execution. We will only provide a short overview of SNAP here; the reader is referred to [9] for more details. SNAP was selected for three of its properties: (1) lightweight execution, (2) convenient service interface, and (3) predictable resource usage.

SNAP is lightweight, as it is designed to require no unmarshalling (and very little or no marshalling) and permits in-place execution and simple memory management. For extensibility purposes, SNAP programs can also access more efficient precompiled node resident *services*. The service interface is plugin-based to allow for runtime addition and removal of services.

A key aspect of SNAP's design is its *predictable* resource usage in terms of CPU time, memory consumption, and network hops. This means that writing a stationary remote agent that "settles in" at a given node is not possible, nor can we write permanently circulating "sentry" agents. Stationary agents are more appropriately designed using existing approaches. Long-running active packets that traverse a network repeatedly can be composed from multiple packets, each of which traverses the circuit a fixed number of times before reporting back to the management station, which then sends out a new packet. A benefit of this scheme is that controlling faulty

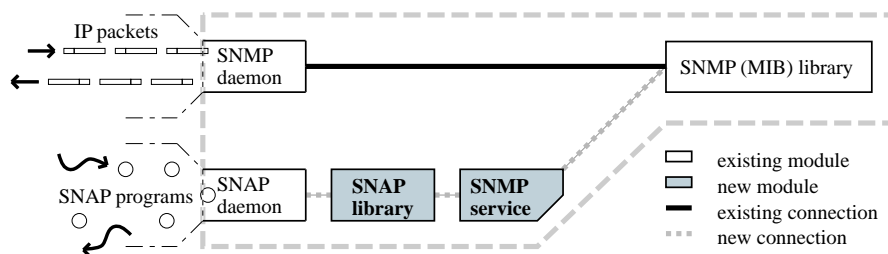


Figure 1: Splash architecture: a common MIB instrumentation (the SNMP library) services requests from both SNMP and mobile agent engines.

agents becomes trivial: the MS simply can simply stop injecting new agents when an error is detected. All outstanding agents will quickly run out of resources and die.

While legacy routers simply forward a SNAP packet toward its destination, SNAP-aware routers will detect its SNAP header and execute it. Packets can be sent directly to the destination or on a hop-by-hop basis. The first method allows for deployment in a largely non SNAP-aware environment. A use of the second will be shown in Section 3.

2.3 Connecting SNAP and SNMP

Rationale

Since we want to compare Splash's performance with that of regular SNMP, we decided to base our Splash agent on the protocol engine of the widely used *net-snmp* (v5.0.6) toolkit. Code interleaving between the two protocol engines was minimized because we (1) have no control over the development of the *net-snmp* code and (2) prefer to have the SNMP access handlers separated from the core SNAP VM. The result is a modular design, as shown in (Figure 1). Maintaining such a system is relatively easy, while performance degradation is minimal.

Adding the SNAP VM to the SNMP agents

By adding a registration function and a protocol engine for mobile agents to *snmpd*, we were able to plug in the SNAP VM without making changes to the *net-snmp* library itself. The SNAP handler function calls have been placed directly around the original agent's `select()` statement, so that SNMP request handling remains unaltered.

Accessing the MIB instrumentation from SNAP

Mobile agents access the MIB instrumentation by creating on the fly SNMP requests that are sent 'in-process' to the SNMP protocol handler. This and other frequently recurring actions have been hardcoded for performance reasons. To distinguish between the SNAP VM and additional services a *plug-in based* service infrastructure has been created which is able to call *any* function implemented in system libraries. All that is needed to export a function to SNAP agents is an explicit registration at compile-time. Figure 1 shows the SNMP service plug-in.

2.4 Network Protocol

We chose to have our clients send their agents to a local SNAP VM, which in turn forwards the agent to its destination using the SNAP infrastructure. This option was preferable from a functionality point of view, since agents can thus take routing decisions autonomously along the way. However, it is also possible to send requests directly to the destination (in UDP packets, just like SNMP), and this would certainly improve SNAP performance. As will be shown in Section 3 these issues are of minor importance, since we are already able to achieve results comparable to standard SNMP. If necessary, however, SNAP can fall back on traditional client/server communication.

3. Experimental Results

While we do not intend to compete directly with the existing SNMP infrastructure for simple requests, we will show that SNAP performance is fairly close to that of SNMP even in this case. Thereafter we will show experimentally that mobile agents can be used to solve what are traditionally considered “hard” problems.

3.1 Experimental Setup

All experiments were carried out on a cluster consisting of twenty Pentium Pro 200 MHz machines running linux 2.4.7, interconnected using standard 100 Mbit Ethernet devices. Each node was connected to two or three others in a honeycomb-like topology. This structure was chosen because it contains no choke points, nodes that would limit the network traversal option space and therefore the quality of the experiments.

We altered the client applications of both net-snmp and SNAP to have them generate performance statistics. We performed each experiment 101 times and present the median. For all tests, the upper and lower quartiles were within 2.57% of the median.

Despite the fact that net-snmp supports SNMPv3 we only used protocol version SNMPv2c for our comparative tests. Version 3 only adds authentication and security features not present in SNMPv2c. Using these technologies has a severe impact on performance. Since SNAP currently lacks such features comparing it with version 3 would be inappropriate.

3.2 Performance

High performance is ensured under all circumstances in Splash by opting for a twofold solution: (1) the Splash agent is fully backward compatible with the standard net-snmp agent and (2) the SNAP mobile agent protocol engine can achieve results comparable to the SNMP engine for all types of requests. Backward compatibility is ensured by minimizing code interleaving, as we explained in Section 2. To justify the second claim we directly compared the performance of SNAP and SNMP for those requests for which SNMP is optimized and mobile agent based architectures are generally deemed too heavyweight: low-level GET and SET transactions.

Test Selection

Because single GET and SET requests represent the most simple request types their performance must be compared. To identify scalability issues aggregated requests have also been included in the test set. Combining requests was accomplished in SNMP by sending a single SNMP packet (PDU) with multiple variable bindings. Since SNAP agents are programs, aggregating results in a single packet was straightforward. An example SNAP packet, the one used in the GET test, is shown in Figure 2.

Performance of GETNEXT, GETBULK and other more complicated request types has not been compared. Applicability of mobile agents to these aggregated request types has been demonstrated before, as discussed in the introduction. Han-

1. travel to destination	3. reverse direction	4. return results
forw	push 1	athome:
bne athome-pc	getsrc	push 7777
2. call SNMP	forwto	demux
push "sysName.0"	pop	5. stack initialization
calls "snmp_getsingle"		#data 0

Figure 2: Example SNAP packet : GET sysName.0

dling simple requests efficiently is a more difficult task and has not been undertaken successfully so far. Therefore we chose to focus solely on their performance.

Performance results are based on the requests depicted in Table 1. To distinguish processing overhead from network transfer time, all requests have been sent to 7 different agents, at increasing distances from the local node. The closest hop (hopcount 1) can be found through the local interface, while each consecutive agent is located 1 network link further away from the client.

Preprocessing

Figure 3 shows the performance of SNAP and SNMP for the various scenarios without ‘preprocessing’, i.e. without the time it takes to setup the environments. When the net-snmp application opens the SNMP library, it initializes several large data structures. For simple requests, this preprocessing can have a large impact on performance. For a single SNMP GET request the overhead is approximately 500 times that of the actual request, while SNAP incurs an overhead of about 30 times the request. With this preprocessing taken into account, SNAP outperforms SNMP by an order of magnitude. Caching of structures can, however, largely hide this weakness. Therefore we chose to keep it out of the direct comparison.

External influences

The results do not reflect the performance of SNAP and SNMP handling alone. External influences that come into play are for instance network transfer, MIB instrumentation processing and data conversion.

We observe that the internal processing time for a request in the MIB is directly dependent on the variable that is being requested. Response times of identical request types but for different variables (e.g. GET sysName.0 and GET ifNumber.0) may vary widely. The factor that determines this is the location of the variable. Requesting kernel values is a more computationally intensive task than copying in-memory variables. We tried to minimize this overhead by requesting values that reside in the local address space.

name	GET	SET	GET5	GETSET
full SNMP request	get (sysName.0)	set (sysName.0)	get (sysName.0, sysContact.0, sysLocation.0, sysServices.0, sysDescr.0)	set (sysName.0), get (sysName.0)

Table 1 Performance Requests

Results

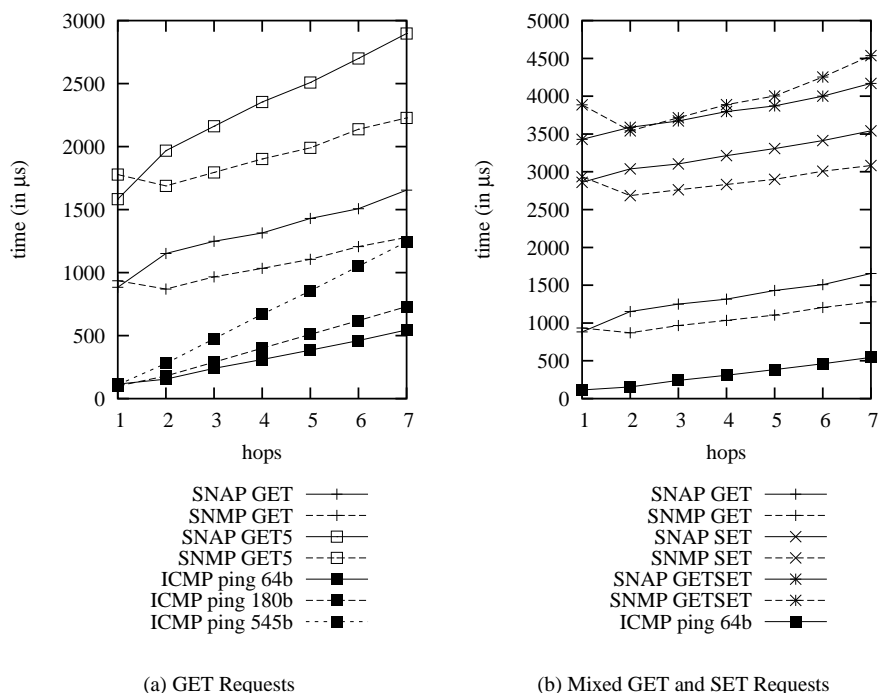


Figure 3: Low-level Requests Round trip Results

Executing a single request using SNAP takes only fractionally longer than executing the same request using SNMP. When sending the request to a remote host the penalty of using SNAP lies between 30% and 10%. The factor evening out the results is network delay. Network delay grows linearly with each hop, as can be seen from the ICMP ping statistics, while remote processing overheads remain the same.

When executing a request at the local host (hopcount 1) the outcome changes dramatically: here SNAP actually outperforms SNMP. From this observation it seems that SNMP consumes more computing resources.

For the combined GET5 request the SNMP PDU outperforms SNAP on all levels. This could be expected, since executing an mobile agent, even a lightweight one, is computationally more intensive than handling a relatively simple SNMP PDU. In principle, the overhead incurred by the GET5 PDU is identical to that of the GET1 PDU, save the processing necessary for the 4 additional variable bindings. What is striking, however, is that the two environments scale differently with hopcount, again at the disadvantage of SNAP. Packet size, even in the limited scope of our research, appears to play a significant role in network delay. For this specific request,

SNMP packets were 101 bytes long when empty and 267 when carrying data. SNAP packets weighed in at 444 and 652 bytes, respectively. A weighed ICMP ping test, also shown in Figure 3, revealed that SNAP GET5 network delays are approximately a factor 2 of their SNMP counterparts.

From the SET request results we can see that altering variables is considerably slower than retrieving them. In both environments this action takes approximately 3 times as much time as a GET. Comparative performance remains similar to the GET case.

Since SNMP cannot combine SET and GET requests in a single PDU two separate packets have to be sent to the remote node. In our experiment the variable that is to be retrieved is the same variable that is being set. Therefore the two have to be carried out in consecutive order. SNAP can combine these requests in a single packet, while SNMP will have to wait for the SET to finish before it can issue the GET. Naturally, SNAP easily outperforms SNMP in such instances. This scenario, while not extremely insightful for low-level performance comparison, gives a first indication of the advantages offered by mobile agents.

Bottlenecks

Additional timed tests for SNAP revealed that not MIB instrumentation access, but network transfer and data conversion are the main bottlenecks. For a single GET, accessing the instrumentation takes up at most 60% of the serverside processing time. When taking into account intermediate processing, as discussed in 2.4, this value drops to below 25%. Since SNAP and SNMP show roughly equal overall performance and instrumentation access is identical for both, we expect similar timings to hold for SNMP.

Conclusion

In general, SNAP outperforms SNMP by 10 to 30 percent. The precise slowdown depends on various factors, such as network delay and request type. Compared to previous results obtained with mobile agent platforms, where performance is often orders of magnitude worse than that of the reference system, this relatively small penalty is a definite improvement. However, a serious drawback we encountered is the increased packet size inherent to using programmable packets and especially the effect this has on network delay. Most important outcome of these tests is that SNAP can execute those requests to which SNMP is especially tailored with only a minor performance penalty. As the SNAP and SNMP results are in the same order of magnitude, SNAP could conceivably be used as a replacement for SNMP. Under normal operation, however, there is no reason not to use Splash's SNMP interface.

3.3 Functionality

Having tested SNAP performance in a head-to-head comparison with ordinary SNMP, we will now exploit the increased flexibility mobile agents offer. The following examples will illustrate how lightweight packets can solve problems that are difficult or even impossible to solve with SNMP alone.

For the presented use-cases we will disregard the issue of performance. The reason for this is simple: using SNMP, it is possible to process large amounts of data

in parallel. Nevertheless, the amount of data that has to be transported may be of such proportions that this is deemed an inapplicable solution to many problems. With the anticipated growth of network-enabled appliances the need for more flexible and resource friendly management tools shall almost certainly increase in the coming years. The introduction of new dynamic networking paradigms (e.g. *ad hoc* networking) especially calls for more intelligent communication [19].

In short, the functionality advantage of mobile agents comes from combining programmability with network roaming capabilities. The overall advantage of having fine-grained control through programmability far overshadows any individual example of this technique. That said, we have drawn up an introductory selection of algorithms that explicitly make use of mobile agents' roaming abilities. For each of the given examples a program has been implemented in Splash. To save space we refrained from presenting the complete programs below. Instead, we show instructive pseudo-code snippets. The complete programs, together with background information can be found on the Splash website at <http://splash-snap.sourceforge.net/papers/noms2004/>.

Surveyor

In [10], a SNAP-based program is discussed that travels to a predefined list of hosts. This application, called a *surveyor*, has the advantage over simple polling that it minimizes the distance traveled through the network and thereby reduces overall management traffic. Figure 4 shows an example of a simple surveyor packet. This surveyor retrieves error information through a service call (the `calls` statement) and forms a per node aggregated error-counter from the individual interfaces' counters.

Traffic savings in this case are twofold. Firstly, bandwidth is saved by aggregating data on the remote hosts. Secondly, by traveling from node to node rather than making repeated round-trips with the management station, the total number of network hops is being reduced. By aggregating totals en route bandwidth needs can be even further reduced.

```

; continue to dest
forw
bne atdest

; retrieve data
push "ifInErrors.2"
calls snmpget
push "ifInErrors.3"
calls snmpget
; aggregate data
add

; move to next host
forwto

atdest: ; return data
demux

```

Figure 4: A simple surveyor packet

```

; has something gone awry?
gti 1000
bez normalrun-pc

; if so, send to a neighbour
specialcase:
getdst
; call a service
calls "if_getallneighbours"
push 0
send

; loop
push specialcase
getdst
forwto

```

Figure 5: Distributed processing

Distributed Surveyor

One of the problems considered hard to perform using SNMP is that of resolving distributed problems. In this section, we consider the example of a Distributed Denial of Services (DDoS) attack. Discovering the originating network nodes and taking action on these nodes is essential to stop the network from becoming flooded. Using SNMP the only way to find out where the problem occurs is by sending messages to a large number (possibly all) of the nodes in the network. This increases the load considerably in an already overloaded situation and, as a side-effect, transmits a lot of data, much of which is useless.

Instead, using Splash, we altered the original surveyor program to react locally when a problem is spotted. At each hop, the network load is compared to a predefined threshold value. If this value is exceeded, normal execution is halted and the packet forwards itself to its neighbours for error detection. By using this algorithm the DDoS test is recursively copied throughout the errorzone and dies out only at nodes that operate under normal load. The management station therefore only receives error reports from those areas that need extra attention.

The resulting code snippet shown in Figure 5 is inserted directly after the operational code in Figure 4. Since SNAP does not allow unlimited loops, we have to locally resend a packet in order to execute a piece of code for an *a priori* unknown number of neighbouring hosts.

Autonomous Surveyor

A more advanced agent, the autonomous surveyor, can travel the network without prior knowledge of the underlying topology and without resorting to extensive route tables. It can alter node behavior by (1) setting interface status (UP or DOWN) to shut down interfaces on nodes from which too much traffic originates and (2) forwarding rules (TRUE or FALSE) to disable forwarding in nodes where error rates are unacceptably high. As *net-snmp* does not directly support this, the corresponding functions were exported from the SNAP service library.

Because this packet can select its next destination from the set of neighbouring nodes, disabling forwarding has no impact on its ability to move through the network. Similarly, as long as we do not disable all of its interfaces, access to the node is retained, even when the routing tables do not reveal this. Such an algorithm can be helpful situations where routing itself is broken. *This type of problem cannot be solved with SNMP.*

Stateful Network

For this last example we added a data dictionary service to demonstrate the advantages of stateful processing. The data dictionary consists of simple set, get and delete instructions for manipulating named variables. Keeping track of previously calculated information on the remote nodes reduces the need for copying intermediate statistics to the monitoring agent. Figure 7 displays the code for comparing an error rate over time. The depicted snippet could be inserted in any one of the packets discussed earlier.

```

forw
; gohome if resources
; are used up
getrb
lti 2
bne gohome-pc

; get and goto a next host
getdst
calls "if_getnexthop"
forwto

; go to client
gohome:
push athome
getsrc
send

; return info
athome:
demux

; calculate new value
push "ifInUcastPkts.2"
calls "snmp_getsingle"
push "sysUptime.0"
calls "snmp_getsingle"
div

; get previous value
push "intraffidx"
calls "memmap_getint"

; push new value
push "intraffidx"
pull 1
calls "memmap_addint"

; compare new and old values
multi 2
gt
bez isok-pc
push "error"

```

Figure 6: autonomous surveyor

Figure 7: Stateful processing

Conclusion

Comparing functionality is harder than comparing performance due to a lack of quantifiable data. We have supplied a number of use-cases where SNMP usage either entails too much overhead or cannot be applied at all. Because of their inherent mobility, the presented algorithms are equally not suited to implementation in more static management by delegation frameworks such as ScriptMIB. We have also demonstrated that these techniques can be implemented easily in Splash through its SNAP protocol engine.

The presented algorithms were kept brief for purpose of brevity, therefore the specific applications are not extremely useful. These programs, however, do not exhaust the mobile agent option space. Other methods for improving network management currently implemented in legacy applications can also be ported to Splash. The obvious benefit is selection freedom: an administrator can trade-off functionality against performance for each individual use-case. As such, Splash can be used to experiment with new algorithms, contrary to SNMP. Furthermore, it is unnecessary to add new applications for each new use-case, as all algorithms can be made to work within this single framework.

4. Related Work

There is a significant body of work concerning mobile agents in network management [1, 3, 5, 11, 22, 12, 13, 15, 17, 24]. Bieszczad *et al.* provide a survey [4] that identifies a number of areas in which mobile agents are applicable to network management.

Existing mobile agent systems [3, 11, 22, 12, 13] geared toward network management tend to be based largely upon Java. Unfortunately, there is no good way to bound the resources consumed by a Java agent, as infinite loops can be expressed. Hawblitzel *et al.* have shown that it is unsafe to simply terminate runaway threads in the JVM [7].

An alternative remote management paradigm is that of management by delegation. Whereas mobile agents are designed to roam freely, delegated agents are meant to settle in at remote locations. Many such schemes have been suggested, e.g. hierarchical management [20]. The only one that appears to have had a measure of success in deployment is the IETF ScriptMIB [8]. The ScriptMIB architecture provides an SNMP-based interface for installing and running scripts, without specifying a particular script programming language. Multiple SNMP round-trips are necessary to set up and invoke a new script (and even more are needed for restarting, revoking, or updating the scripts, or even determining whether or not they have terminated). In contrast, SNAP-based agents are more “light on their feet,” being self-contained. For this reason also, the ‘management of management’ that was found to be ‘a significantly complicated task’ in ScriptMIB [23], is virtually non-existent in Splash. Moreover, the current use of languages like Java and Tcl in ScriptMIB may result in safety problems, as it is difficult to restrict their resource consumption (such as CPU or memory usage). SNAP agents are intrinsically resource safe and require no additional safety mechanisms. Meanwhile, RFC3179 describes SMX, the extensibility protocol for ScriptMIB[16]. In SMX, the communication between between scripts (in virtual machines) and the core management agent takes place as inter-process communication over simple TCP connections. While this makes it easy to extend the framework with new VMs, it is not quite as efficient as a direct library call within the same address space, which is used by Splash. While we think it is *likely* that ScriptMIB implementations will follow RFC3179, we should point out that strictly speaking this is not *mandatory*. Summarizing, ScriptMIB requires explicit signaling both to install and activate scripts (or install the schedule according to which they should be activated), while the SNAP packets function more like active packets (or *mobile* agents): picking their way through the network, while being executed at the appropriate nodes. The price for these advantages, however, is that it is impossible to use Splash for true management by delegation on the managed nodes. For this reason, we see the two approaches as complementary.

Perhaps the most closely related project to ours is the Smart Packets project from BBN [18]. Indeed, their system closely resembles ours in having a bytecode interpreter for active packets. The main difference from our work is that Smart Packets cannot direct themselves; they are sent from a source to a destination, and may execute on intermediate nodes, but may not deviate from the original path. As a result, Smart Packets do not offer quite the agility needed for truly mobile agents. Secondly, Smart Packets use instruction counters and memory limits to monitor and restrict resource usage, whereas the SNAP programming language provides a *de facto* guarantee of well-behavedness. Finally, no performance data is available to determine whether the Smart Packets execution environment is lightweight or not.

5. Conclusions and Future Work

We have presented an architecture for adding mobile agent support to standard SNMP. This architecture minimizes the overlap of the code bases for the underly-

ing `snmpd` and the mobile agent platform while providing the functionality of both within a single process. Our implementation of this architecture, Splash, combines the widely-deployed `net-snmp` package with the SNAP mobile agent environment. The hybrid nature of Splash allows it to circumvent the main drawback of other SNMP alternatives, i.e. lack of interoperability.

Experimental results verify that (1) Splash can, contrary to previous mobile agent based management tools, execute low-level request at roughly the same speed as SNMP, while (2) at the same time reduce management traffic and improve overall responsiveness vs. SNMP in use-cases where derived or distributed results come into play. Allowing for both low-level SNMP requests and modern, more elaborate, algorithms, Splash gives network administrators the option to select the solution most suitable to any given situation. By doing so, Splash creates a safe environment for adoption of new networking practices. We have released Splash in the public domain and a trial project of the technology in a metropolitan area wireless network has started.

Acknowledgments

Herbert Bos wishes to acknowledge the support of the Leiden Institute of Advanced Computer Science at the Universiteit van Leiden, The Netherlands, with which he was affiliated prior to moving to the Vrije Universiteit in Amsterdam. Jon Moore's work was supported by the NSF under Contracts ANI 00-82386 and ANI 98-13875.

References

- [1] M. Baldi, S. Gai, and G. Picco. Exploiting Code Mobility in Decentralized and Flexible Network Management. In *Proceedings of the First International Workshop on Mobile Agents*, pages 13–26, April 1997.
- [2] M. Baldi and G. P. Picco. Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications. In *Proceedings of the 20th International Conference on Software Engineering (ICSE'98)*, pages 146–155, April 1998.
- [3] C. Baumer and T. Magedanz. The Grasshopper Mobile Agent Platform Enabling Short-term Active Broadband Intelligent Network Implementation. In *IWAN'99*.
- [4] A. Bieszczad, T. White, and B. Pagurek. Mobile Agents for Network Management. *IEEE Communications Surveys*, 1(1), September 1998.
- [5] Markus Breugst and Thomas Magedanz. Mobile Agents—Enabling Technology for Active Intelligent Network Implementation. *IEEE Network*, 12(3):53–60, May/June 1998.
- [6] W. Eaves, L. Cheng, A. Galis, T. Becker, T. Suzuki, S. Denazis, and C. Kitahara. SNAP Based Resource Control for Active Networks. In *GLOBECOM'02*.
- [7] C. Hawblitzel, C. Chang, G. Czajkowski, D. Hu, and T. von Eicken. Implementing Multiple Protection Domains in Java. In *Proceedings of the 1998 USENIX Annual Technical Conference*, June 1998.
- [8] D. Levi and J. Schoenwaelder. Definitions of Managed Objects for the Delegation of Management Scripts. RFC 3165, IETF, August 2001.
- [9] J. T. Moore. *Practical Active Packets*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, July 2002.

- [10] J. T. Moore, J. Kornblum Moore, and S. Nettles. Predictable, Lightweight Management Agents. In *IWAN'02*, December 2002.
- [11] Objectspace, Inc. Voyager Application Server 4.0 Datasheet, October 2000.
- [12] A. Puliafito and O. Tomarchio. Using Mobile Agents to Implement Flexible Network Management Strategies. *Computer Communications Journal*, 23(8):708–719, April 2000.
- [13] D. Raz and Y. Shavitt. An Active Network Approach for Efficient Network Management. In *IWAN'99*, Berlin, May 1999.
- [14] M. G. Rubinstein and O. C. Duarte. Evaluating Tradeoffs of Mobile Agents in Network Management. *Networking and Information Systems Journal*, 2(2):237–252, 1999.
- [15] A. Sahai, C. Morin, and S. Billiart. Intelligent Agents for a Mobile Network Manager. In *Proceedings of the IFIP/IEEE International Conference on Intelligent Networks and Intelligence in Networks (2IN'97)*, September 1997.
- [16] J. Schoenwaelder and J. Quittek. Script MIB Extensibility Protocol Version 1.1. RFC 3179, IETF, August 2001.
- [17] C. Schramm, A. Bieszczad, and B. Pagurek. Application-Oriented Network Modeling with Mobile Agents. In *NOMS'98*, February 1998.
- [18] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge. Smart Packets: Applying Active Networks to Network Management. *ACM Transactions on Computer Systems*, 18(1):67–88, February 2000.
- [19] C. Shen, C. Srisathapornphat, and C. Jaikaeo. Adaptive Autonomous Management of Ad hoc Networks. In *NOMS'02*, Florence, Italy, April 15–19 2002.
- [20] M. R. Siegl and G. Trausmuth. Hierarchical network management: a concept and its prototype in SNMPv2. *Computer Networks and ISDN Systems*, 28(4):441–452, 1996.
- [21] P. Simes, J. Rodrigues, L. Silva, and F. Boavida. Distributed retrieval of management information: Is it about mobility, locality or distribution? In *NOMS'02*, Florence, Italy, April 15–19 2002.
- [22] P. Simões, L. M. Silva, and F. Boavida-Fernandes. Integrating SNMP into a Mobile Agent Infrastructure. In *DSOM'99*, October 1999.
- [23] Frank Strausz. Advantages and disadvantages of the script mib infrastructure, scriptmib project report, october 2000.
- [24] G. Susilo, A. Bieszczad, and B. Pagurek. Infrastructure for Advanced Network Management based on Mobile Code. In *NOMS'98*, February 1998.
- [25] T. Thorn. Programming Languages for Mobile Code. *ACM Computing Surveys*, 29(3), September 1997.