Policy-driven Network Simulation: a Resilience Case Study

Alberto Schaeffer-Filho
School of Computing and Communications
Lancaster University,
Lancaster, UK
asf@comp.lancs.ac.uk

Paul Smith
School of Computing and Communications
Lancaster University,
Lancaster, UK
p.smith@comp.lancs.ac.uk

Andreas Mauthe
School of Computing and Communications
Lancaster University,
Lancaster, UK
andreas@comp.lancs.ac.uk

ABSTRACT

Networks must be resilient to challenges such as malicious attacks or network overload and adapt their operation in an autonomous manner. Network simulations enable the testing of complex network scenarios (which would be difficult to emulate using actual hardware) in an inexpensive manner. However, it is difficult to evaluate resilience strategies that involve the interplay between a number of detection and remediation mechanisms that must be activated on demand according to events observed in the network (as opposed to hardcoded protocols). In this paper we propose the notion of a *policy-based resilience simulator* based on the integration of a network simulator and a policy management framework. This permits the evaluation of resilience strategies consisting of mechanisms whose behaviour can be adapted during run-time – e.g. setting flags, dropping connections, triggering or stopping monitoring sessions, etc. We employ policies to specify the required adaptations, which are de-coupled from the hard-wired implementations of the simulated components, according to conditions observed during run-time in the simulation. We can thus observe how real policies affect the operation and the behaviour of simulated components, and then evaluate the effectiveness of resilience strategies before they are deployed in the network infrastructure.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Experimentation

Keywords

Network simulation, resilience, policy management, adaptation, survivability

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'11 March 21-25, 2011, TaiChung, Taiwan.
Copyright 2011 ACM 978-1-4503-0113-8/11/03 ...$10.00.

1. INTRODUCTION

Next generation networks must be resilient to challenges, such as malicious attacks and operational overload, and adapt their operation accordingly. Resilience is the ability of the network to maintain acceptable levels of operation in the face of a challenge [21]. Resilience strategies typically involve a complex interplay between detection and remediation mechanisms that must be activated on demand according to events observed in the network. For example, a number of detection mechanisms could be progressively employed to construct an understanding of the root cause of a challenge, which could in-turn lead to the invocation of more-specific and improved remediation mechanisms. Furthermore, these mechanisms may span multiple autonomous systems, requiring careful configuration. In order to define such configurations of mechanisms for resilience we have advocated the use of policies [19]. Policies can separate the hard-wired implementation of network components from their management strategy, and thus permit the easy modification of a strategy without requiring changes in the underlying implementation of the system. This ability to flexibly define a resilience strategy is a necessity, considering the changing nature of challenges that a network may face over time, such as new forms of attacks.

An approach to evaluate network operation, mechanisms and protocols before they are deployed in the network is to conduct simulations that indicate the impact on the behaviour of network components. However, it is difficult to evaluate complex resilience strategies that configure and manage multiple mechanisms dynamically subject to conditions observed in the network during run-time (as opposed to hardcoded protocols). Borne from a requirement to evaluate policy-based resilience strategies (which are expressed using policies that configure resilience mechanisms) we have developed a toolset that couples policy management and network simulation. Policies specify the required adaptations based on conditions observed during run-time in the simulation itself. This toolset allows us to assess how different sets of resilience policies affect the operation and performance of the simulated components dynamically.

The primary contribution of this paper is to present a way how to integrate a policy framework and network simulation, and based on this introduce the toolset we have developed. One of the direct benefits of integrating a network simulator with a policy framework is that we can understand how real policies affect the operation of resilience mechanisms running within the simulation environment, and then evaluate resilience strategies before they are implemented and de-

ployed in the network infrastructure (e.g. in routers). This enables the evaluation of complex resilience strategies without the need of a real testbed deployment of mechanisms, which typically involves high costs of hardware and effort. Finally, such a toolset is also necessary due to the possible scale of the types of challenges we want to simulate, which tend to affect multiple autonomous systems, and that would be difficult to reproduce using a real testbed.

This paper is structured as follows: Section 2 describes our approach for policy-based resilience. Section 3 discusses the most important design decisions in the policy-driven network simulator. Section 4 outlines our requirements and the network simulators available, and describes our implementation. Section 5 describes a case-study evaluation of the prototype. Finally, Section 6 presents the concluding remarks.

2. POLICY-BASED RESILIENCE

Although a wide range of resilience mechanisms can be used, such as traffic classifiers and intrusion detection systems, these may span multiple autonomous systems and it is often not clear how they should be orchestrated to realise resilience requirements and adapt their behaviour dynamically. Moreover, a number of mechanisms could be progressively employed to construct an understanding of the state of the network, for example, using multi-stage detection of traffic anomalies, in order to invoke a more-specific and tailored remediation mechanism. The use of policies to define configurations of mechanisms that can ensure the resilience of networked systems has been previously advocated in [19].

2.1 Resilience Mechanisms

Effective strategies for resilience require appropriate coordination of detection and remediation mechanisms. *Detection mechanisms* are used to identify the occurrence of a challenge. Characteristics of the challenges to be detected determine which mechanisms can be used, and where they should be placed (e.g., in border routers or close to potential attack targets). More than one detection algorithm could be used, for example, having an always-on, low-cost anomaly detection algorithm, alongside an offline, more computationally expensive traffic classifier to diagnose the root cause of an anomaly. *Remediation mechanisms* can be activated in response to detected challenges. They are intended to maintain acceptable levels of service in the presence of challenges. Example remediation mechanisms can include *Pushback* [10] of bad traffic (e.g. DDoS attacks) and rate limiting overwhelming but legitimate traffic (e.g. flash crowds).

Simulating resilience strategies requires the implementation of a range of plausible challenges [3], and corresponding detection and remediation mechanisms. Reference implementations for a number of anomaly detection and classification algorithms described in the literature are available in the Java-ML toolkit [8]; we expect to integrate these mechanisms in our simulation models.

2.2 Policy Management

Policies allow the de-coupling of the hard-wired implementation of system components from their management strategy. This permits the modification of the management strategy of networked resources during run-time. *Event triggered condition-action* (ECA) rules can be used to define the adaptation strategy of the system [18]. For example, the policy illustrated in Figure 1 can be used to reconfigure the behaviour of a rate limiting mechanism based on the occurrence of a *high risk* event (raised, for example, by an anomaly detection component) and additional contextual information, e.g. current link utilisation. Instead of having the management strategy hardcoded into network components, policies separate the management from the component implementation. By having detection and remediation mechanisms in the simulation controlled by policies, we can specify their management strategy dynamically.

```
1 on AnomalyDetectorMO.highRisk (link, src, dst)
2     if (LinkMonitorMO.getUtilisation() >= 75%)
3         do RateLimiterMO.limit(link, 60%)
```

Figure 1: Policy configuring rate limiter upon high risk event (pseudo-syntax).

Policies are written in terms of *managed objects* (MOs). The policy in the example above requires three MOs: *(a)* an *anomaly detector* MO implementing a detection algorithm, which raises a *highRisk* event when an anomaly is identified (providing details such as the *link* where the anomaly was detected, and the *src* and *dst* addresses associated with the anomalous IP flow); *(b)* a *link monitor* MO which can be used to obtain further information about the state of the network, such as current utilisation; and *(c)* a *rate limiter* MO which can be dynamically configured to mitigate the anomaly, e.g. limiting the link capacity while the cause of the anomaly is investigated (which can be done by more expensive classification algorithms). Each MO must provide a *management interface*, which defines the scope for the specification of the policies that control the corresponding object.

3. DESIGN ISSUES

Our work is based on the integration of two standard toolsets in the area of network research: a *network simulator*, which permits the evaluation of networks and protocols under specific test scenarios, and a *policy management framework*. We discuss in the following the main issues in the design of the resilience simulator.

3.1 Integration Techniques

In [11], a number of techniques to allow the integration between a network simulator environment and external third party applications are discussed:

***Socket connection*:** proxies within the simulation maintain socket connections to third party applications, which can be easily integrated without changes to the third party application. However, this technique may cause CPU scheduling issues since simulations run faster and consume more CPU. Synchronisation issues may become a problem because the simulation and the application run in their own time domain (simulation time is typically faster than real-time).

***Source code integration*:** code integration is straightforward for simple applications and cause no time distortions. However, this may be technically difficult for larger applications because of dependencies that must be resolved in the build environment. Moreover, threads in the third party application may still suffer from CPU scheduling issues and cause problems such as access violations.

***Shared libraries*:** is based on the integration between the simulation tool and the binary code of the third party application. It is similar to source code integration but avoids problems related to the building process, because the build environments for the simulator and for the third party application are kept separated. It still suffers from the threading and timing problems.

Our integration is based on proxies, similar to the *socket connection* method, but using RMI objects instead (discussed in Section 4). Typically, this technique can be used if the third party application does not need data from lower layer protocols [11]. We expect that CPU scheduling and synchronisation issues can be mitigated because, in contrast to the applications in [11], we do not exchange packet-level information (large quantity, fast processing) with the policy framework. Instead, exchanges are limited to selected control events and corresponding management commands, and thus synchronisation issues are reduced. Alternatively, a new event scheduler may be implemented to slow down the simulation speed to real time, as in [1].

To mitigate CPU scheduling problems, a higher process priority may be assigned to the third party application as suggested in [11], which is also much easier to achieve compared to changing priorities of specific threads as required by the other techniques[1]. Despite the restrictions associated with the socket technique, it allows us to quickly integrate the policy framework and the simulation environment.

3.2 Adaptive Simulation Behaviour

Our methodology for the specification of adaptive simulations is divided into the following steps: firstly, purpose-specific detection and remediation components with their corresponding callback functions must be implemented so they can be used alongside standard simulation objects, e.g. a library of remediation mechanisms may include a custom *rate limiter* component, as well as a customised *Webserver* component implementing a number of adaptive actions. Secondly, these components running in the simulation must export their callback functions through management interfaces accessible to external applications, e.g. a policy decision point (PDP). Thirdly, communication between simulated objects and the external policy framework must be implemented via adapter objects, which need to abstract invocation details (e.g. using sockets, RMI, RPC) and forward commands to objects inside the simulation. Fourthly, an event broker is needed to resolve and forward event notifications from inside the simulation to the policy engine. Finally, policies are used to define resilience strategies by determining which adaptive actions must be invoked in response to events raised in the simulation. The overall architecture is illustrated in Figure 2.

4. IMPLEMENTATION

We created an environment to evaluate policy-based resilience strategies such as those in [19]. Our prototype is based on the integration of a network simulator and the Ponder2 framework [22].

[1]We leave aside issues related to access violations and concurrency; in the examples we have been working on, policies cause only atomic changes, e.g. setting flags, which do not incur access violations. In more complex examples, we may need to circumvent this problem through artefacts of a specific tool, such as OMNeT++'s macros `Enter_Method` and `Enter_Method_Silent`, or `synchronized` blocks in SSFNet.

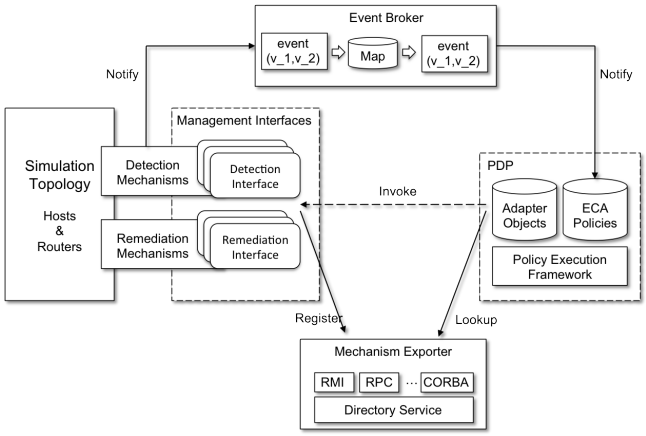

Figure 2: Resilience simulator architecture.

4.1 Candidate Network Simulators

For the policy-based resilience simulator, we have considered the use of the most popular network simulators, including NS-2 [13], NS-3 [14], OMNeT++ [15], SSFNet [20] and OPNET [16]. The choice of a network simulator is constrained by a number of requirements. Firstly, the simulator needs to be extended and instrumented, not only in terms of protocol models, but also through the implementation of the necessary communications to allow the policy framework to interface with the simulation. Secondly, the availability of a large number of network models will allow faster modelling of networks, and their resilience strategies. Thirdly, the simulator should also have good scalability, speed and performance to allow faster and larger simulations. Finally, we are interested in experimenting with multi-level approaches to resilience, consisting of mechanisms for resilience that will reside in layers 1-7. Therefore, the simulator should be general, and allow the modelling of communication networks, distributed, parallel systems, as well as P2P networks.

Since our goal is to model resilience in general communication networks, purpose-specific tools such as the WSN simulator TOSSIM [9] are not considered. OPNET is a commercial tool and the source code of its simulation kernel is not publicly available. Since we are required to extend the simulator to facilitate the communication with the policy framework, we abandoned this option. Likewise, NS-2 has been consistently reported to have poor scalability and a large memory footprint compared to other simulators [2, 24], and for this reason we discarded this option as well. The remaining candidates are discussed:

***NS-3*:** uses C++ and can be optionally combined with Python scripts. NS-3 is a major revision of NS-2, which is targeted for scalability, extensibility, modularity, emulation and clarity of design, focusing on layers 2-4 of the protocol stack [7]. However, NS-3 lacks an extensive library of models, and NS-2 models need to be ported to NS-3 manually. One of the goals of NS-3 is to support simulations that integrate well with virtual machines, network testbeds and implementation code [7]. NS-3 is reported to have a good performance [24].

***OMNeT++*:** uses C++ to model behaviour and NED (Network Description Language) to describe topology. Unlike NS-2, these are highly decoupled. To enable large simulations, models are hierarchical and built from reusable components [23]. OMNeT++ has a modular, extensible architecture, in which multiple simple modules, e.g. protocols, are combined into a compound module, e.g. host node. OMNeT++ allows a limited amount of dynamic behaviour in the simulations through *parametric topologies*, e.g. number of nodes can be reconfigured during runtime. OMNeT++ is reported to have a good overall performance [24].

***SSFNet*:** is a standard for discrete-event simulation with implementations in C++ and Java. SSFNet uses DML for topology description and configuration, which is a text-based format similar to XML. DML is equivalent to OMNeT++'s NED, however, according to [23], DML has less expressiveness and features to support large-scale models built from reusable components. In [12], the C++ implementation of SSFNet is reported to present a good compromise between execution speed and memory requirements when compared to other simulators. However, development of the SSFNet simulator and models was discontinued in 2004.

We considered NS-3, OMNeT++ and SSFNet equally suitable for our requirements. Due to our previous experience with SSFNet and our familiarity with its API, including an API for setting up DDoS attack scenarios[2], the implementation presented in this section is based on this simulator. However, we will port this prototype to an OMNeT++ implementation as part of our future work, since OMNeT++ is considered one of the most used simulators for research in the area of communication networks [4].

4.2 Ponder2 Policy Framework

Ponder2 comprises a general-purpose object management system. It implements a policy execution framework that supports the enforcement of ECA and authorisation policies [22]. Policies are written in terms of *managed objects*, which are stored in a local domain service. Ponder2 provides built-in *factories* for the creation of core managed objects, however, the infrastructure is extensible and allows the creation of user-defined managed objects, e.g. *adaptors* for interfacing with remote resources. Managed objects are programmed in Java. Several protocols are supported to facilitate communication with remote resources, e.g. RMI, HTTP[3]. A command interpreter supports a high-level configuration language called *PonderTalk*, which allows the invocation of actions on these managed objects. Based on our previous investigations [17], Ponder2 was considered to be more extensible and with better infrastructure support when compared to other policy frameworks.

4.3 Prototype Implementation

SSFNet is used for modelling and simulation of Internet protocols and networks at and above the network layer. Protocols are composed hierarchically to define components in the network. Each protocol is implemented by a class which extends SSFNet's `ProtocolSession`. DML files are used to define how protocols are composed for each network component. Resilience mechanisms are instrumented versions of classes extending `ProtocolSession` at different layers of the protocol stack. The instrumentation depends on the mechanism and protocol layer, e.g. a `HttpServerRemediation` class may offer callbacks for redefining the *response delay* for the server, whereas a `RateLimiter` may offer callbacks for changing the *bit rate* limit for a router.

We defined an abstract RMI interface that must be implemented by instrumented objects to indicate that they offer callbacks. A further management interface extends the abstract interface and adds to it the management operations that each mechanism supports. This interface is exported by a `MechanismExporter`, which implements a directory service and allows the policy framework to perform a directory lookup and obtain a reference to selected simulated objects. Simulated objects may also notify the policy framework of changes in the state of the simulation via `Ponder2Broker`, which resolves events generated within the simulation to Ponder2 events. A custom `RemediationAdaptor` was implemented to facilitate the communication between Ponder2 and SFFNet objects. Each instance of `RemediationAdaptor` functions as a RMI proxy for an actual simulated object. The implementation of the actions defined in a management interface is the responsibility of the actual objects via the instrumentation of the corresponding classes in the simulator. Figure 3 outlines the operation of our prototype[4].

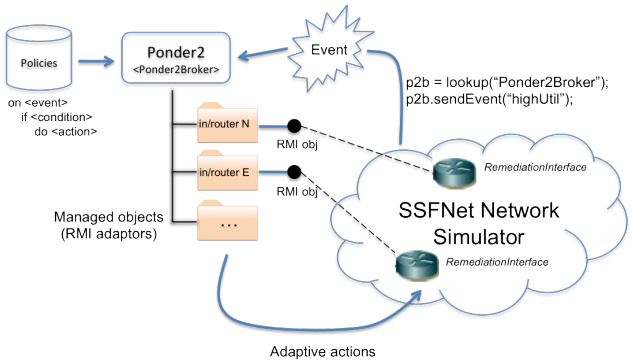

Figure 3: Integration between SSFNet and Ponder2.

Figure 4 shows an example policy named `adaptHigh` which is triggered by the event `highUtil` and invokes an adaptive action on a `server` object, if the utilisation is greater or equal to 75% (`value` is one of the parameters of the event `highUtil`). The event can be generated by objects in the simulation monitoring the utilisation of a particular link, and published in the policy framework via `Ponder2Broker`. When the event triggers the policy, the action `setResponseDelay` is invoked on the `server` object. Both the condition and the action are written as Ponder2 *blocks*[5]. Commands sent to `server` are forwarded to the corresponding SSFNet object executing in the simulation.

[2]http://www.ssfnet.org/javadoc/SSF/App/DDoS/package-summary.html

[3]Adaptors also simplify the integration between the Java-based implementation of Ponder2 and C++ implementations of simulators such as OMNeT++.

[4]Note that multiple instances of Ponder2 can be used if more than one policy decision point (PDP) is necessary.

[5]A *block* is a Ponder2 construction that defines one or more statements (within square brackets) whose execution can be delayed until it is decided that the block should be evaluated – e.g. when the policy is triggered by the event.

```
1 //event template definition
2 highUtil := factory/event create: #( "value").
3
4 //policy definition
5 adaptHigh := factory/ecapolicy create.
6 adaptHigh event: event/highUtil.
7 adaptHigh condition: [ :value | value >= 75].
8 adaptHigh action: [server setResponseDelay:500.].
9 adaptHigh active: true.
```

Figure 4: Policy configuring response delay on server object (PonderTalk syntax).

5. EVALUATION

We demonstrate our prototype through a case-study that shows how a traffic pattern that saturates the link capacity can be controlled by policies adapting the behaviour of objects within the simulation. The topology is illustrated in Figure 5, and is based on an example topology distributed with SSFNet. It consists of a TCP two-stage client/server network in which clients belong to one of two subnets, and send requests to servers in one of 10 subnets (which contain 20 servers each). Subnet 0 is of particular interest, and contains 10 clients connected to a router via 100 Mbps links. This router is configured to monitor and export `netflow` data using the built-in module `SSF.OS.ProbeSession`. Random delay between 0 and 5 ms is associated with each link. Each client makes one request of approximately 1 GB from a given server. Clients in another subnet (subnet 1001) generate additional traffic.

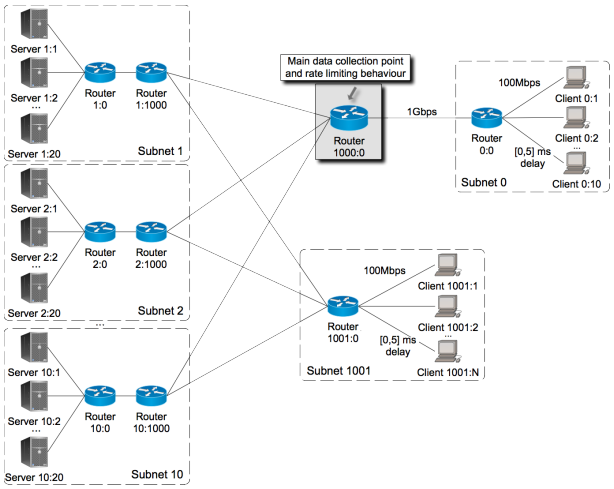

Figure 5: Topology specification for the case-study.

The backbone router with a 1 Gbps link capacity indicated in the topology (`router 1000:0`) is instrumented with a rate limiting mechanism and is the focus of this case-study. We developed a rate limiter for SSFNet that allows the dynamic change of the maximum bit rate for a given router interface. The `RateLimiter` implements a management interface that defines the method `setBitrate(Float factor)`, which multiplies the allowed bit rate for the router by the `factor` given as parameter. The simulated object is registered as a RMI object and a reference to it is exported to Ponder2.

Policy-based rate limiting is defined in terms of *PonderTalk* policies similar to the one in Figure 4. Although we manually generated events signalling high link utilisation, we intend to use implementations such as the one in [4] for the *detection* of anomalies in the simulation, thus allowing us

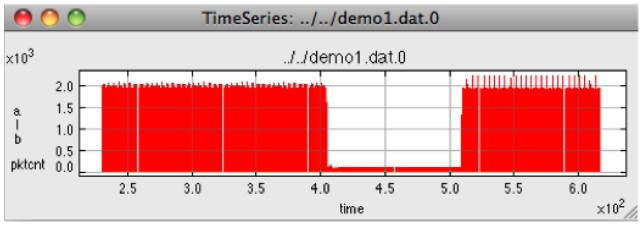

Figure 6: Effects of rate limiting are visible through packet counts monitored at the instrumented router.

to focus on the *remediation* mechanisms. When policies are triggered, they evaluate the events and adapt the operation of `router 1000:0` by invoking `setBitrate(Float factor)`.

Figure 6 illustrates the effects of policy-based rate limiting. We plot the packet counts monitored at the instrumented router. We activated the rate limiting shortly after 4.0×10^2 seconds of simulated time. Its effect is a cap in the maximum bit rate supported in the network interface of the router, reducing packet counts from about 2.0×10^3 packets to 0.1×10^3 packets[6]. Finally, a new event signals that the adaptive mechanism should be withdrawn, thus triggering a second policy shortly after 5.0×10^2 seconds that reverts the router to its original bit rate.

This illustrates how instrumented objects in the simulation can be dynamically configured by management policies. However, the toolset is meant for the evaluation of more complex scenarios, consisting of the coordination between a large number of mechanisms (e.g. rate limiter, flow exporter, link monitor, anomaly detector, traffic classifier, etc) employed in typical policy-based resilience strategies [19].

6. CONCLUDING REMARKS

In this paper, we leveraged our expertise in network resilience and policy-based management, and presented a model for the integration of a policy framework and a network simulator. The developed toolset can be used for the simulation and evaluation of policy-based resilience strategies. We are interested in simulating strategies for managing a number of mechanisms that must cooperate to enforce the resilience of the network in response to operational challenges.

Policies permit the modification of a management strategy (e.g. setting flags, dropping connections, changing routes, adding extra delay to packets, triggering or stopping monitoring, etc) based on conditions observed during run-time in the simulation. Based on the integration between a network simulator and a policy framework, we can thus observe how real policies affect the operation and the behaviour of the resilience mechanisms in the simulation, and then understand and evaluate the effectiveness of multiple resilience strategies before they are implemented and deployed in the network infrastructure. However, the notion of *policy-driven network simulation* is not restricted to the evaluation of resilience strategies only, and the same principles could be used to evaluate other aspects of network management (e.g. using policies to dynamically evaluate changes to QoS parameters of specific components in the simulated network).

The work in [4] also advocates the integration between a network simulator (OMNeT++) and external tools: in particular, *ReaSE* [6], a tool for generation of internet-specific

[6]Note that in order to highlight the activation of the mechanism we multiplied the bit rate by a very small factor.

topologies, background traffic and attack traffic, and *Distack* [5], a framework for attack detection that allows the implementation of various detection methods. The integrated toolchain was used for large-scale evaluation of distributed attack detection. Our work is complementary to that, and while the authors of [4] focused on the integration of tools for *detection* of anomalies in the simulated network, we focus on the integration between a network simulator and a policy framework to enforce the *remediation* of those anomalies.

The prototype presented in this paper is based on a SSFNet implementation and permitted us to assess the viability of the integration. However, as future work we will port it to OMNeT++, which is one of the most popular network simulators in this research area. We expect that due to our choice of a method of integration based on socket connections, porting the implementation to the new simulation platform will be greatly simplified.

7. ACKNOWLEDGMENTS

The work presented in this paper has been supported by the EPSRC funded India-UK Advance Technology Centre in Next Generation Networking. This work has also been supported by the European Commission, under Grant No. FP7-224619 (ResumeNet project). The authors would like to thank Azman Ali for the implementation of the rate limiter used in the evaluation of our prototype.

8. REFERENCES

[1] I. Baumgart, B. Heep, and S. Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007*, pages 79–84, Anchorage, AK, USA, May 2007.

[2] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of manet simulators. In *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 38–43, Toulouse, France, October 2002. ACM.

[3] E. K. Çetinkaya, D. Broyles, A. Dandekar, S. Srinivasan, and J. P. Sterbenz. A Comprehensive Framework to Simulate Network Attacks and Challenges. In *RNDM'10: Second International Workshop on Reliable Networks Design and Modeling*, Moscow, Russia, October 2010.

[4] T. Gamer and C. P. Mayer. Large-scale evaluation of distributed attack detection. In *Simutools '09*, pages 1–8, Rome, Italy, March 2009. ICST.

[5] T. Gamer, C. P. Mayer, and M. Zitterbart. Distack – a framework for anomaly-based large-scale attack detection. In *SECURWARE '08: Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies*, pages 34–40, Cap Esterel, France, August 2008. IEEE CS.

[6] T. Gamer and M. Scharf. Realistic simulation environments for ip-based networks. In *Simutools '08*, pages 1–7, Marseille, France, March 2008. ICST.

[7] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. ns-3 project goals. In *WNS2 '06: Proceeding of the 2006 workshop on ns-2: the IP network simulator*, Pisa, Italy, October 2006. ACM.

[8] Java-ML Website. Java Machine Learning Library (Java-ML). http://java-ml.sourceforge.net/. Accessed March 2010.

[9] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137, Los Angeles, California, USA, November 2003. ACM.

[10] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev.*, 32(3):62–73, 2002.

[11] C. P. Mayer and T. Gamer. Integrating real world applications into OMNeT++. Telematics Technical Report TM-2008-2, Institute of Telematics, Universität Karlsruhe (TH), Feb. 2008.

[12] D. M. Nicol. Scalability of network simulators revisited. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference*, Orlando, FL, January 2003.

[13] NS-2 Website. The Network Simulator - NS-2. http://www.isi.edu/nsnam/ns/. Accessed March 2010.

[14] NS-3 Website. The NS-3 network simulator. http://www.nsnam.org/. Accessed March 2010.

[15] OMNeT++ Website. OMNeT++. http://www.omnetpp.org/. Accessed March 2010.

[16] OPNET Website. OPNET Modeler Accelerating Network R&D (Network Simulation). http://www.opnet.com/solutions/network_rd/modeler.html. Accessed March 2010.

[17] A. Schaeffer-Filho. *Supporting Management Interaction and Composition of Self-Managed Cells*. PhD thesis, Imperial College London, 2009.

[18] M. Sloman and E. Lupu. Security and management policy specification. *IEEE Network*, 16(2):10–19, Mar.-Apr. 2002.

[19] P. Smith, A. Schaeffer-Filho, A. Ali, M. Scholler, N. Kheir, A. Mauthe, and D. Hutchison. Strategies for network resilience: Capitalising on policies. In *4th International Conference on Autonomous Infrastructure, Management and Security (AIMS)*, pages 118–122, Zurich, Switzerland, June 2010. LNCS.

[20] SSFNet Website. Modeling the Global Internet. http://www.ssfnet.org/. Accessed March 2010.

[21] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Comput. Netw.*, 54(8):1245–1265, 2010.

[22] K. Twidle, E. Lupu, N. Dulay, and M. Sloman. Ponder2 - a policy environment for autonomous pervasive systems. In *POLICY '08: Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks*, pages 245–246, Palisades, NY, USA, June 2008. IEEE Computer Society.

[23] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Simutools '08*, pages 1–10, Marseille, France, March 2008. ICST.

[24] E. Weingärtner, H. vom Lehn, and K. Wehrle. A performance comparison of recent network simulators. In *ICC 2009: IEEE International Conference on Communications*, Dresden, Germany, June 2009.