Proceedings of the 2002 IEEE
Workshop on Information Assurance and Security
United States Military Academy, West Point, NY, 17–19 June 2002

Middleware Performance Data-Based Methodology for Information Assurance

Swapna S. Gokhale
Dept. of CSE
University of Connecticut
Storrs, CT 06269
ssg@engr.uconn.edu

Aniruddha S. Gokhale
Dept. of EECS
Vanderbilt University
Nashville, TN 37235
a.gokhale@vanderbilt.edu

Jeffrey G. Gray
Dept. of CIS
University of Alabama
Birmingham, AL 35294
gray@cis.uab.edu

Abstract—

Rapid advances in hardware, networking and software technologies are constantly reshaping the structure and content of the cyber infrastructure by offering newer and powerful tools and services to the end users. However, this infrastructure remains prone to massive disruptions caused by both benign failures including node, link and software failures, and by malicious attacks including denial of service and intrusions. Since no prevention strategy can completely eliminiate such disruptions, timely detection of the occurrence of such disruptions can help mitigate their impact. Existing detection mechanisms based on data collected at lower layers are inadequate because they lack semantic information. There is thus an increasing need to base disruption detection on data that is semantically rich, yet generic enough so that the detection strategies based on these data can be applied uniformly across a wide range of services. Middleware, with its reusable building blocks, which has been the key enabler in the development of cyber services can provide semantically rich data in a generic manner across several services. Since cyber services need to have an acceptable performance in order to be useful, in this paper we advocate the use of performance metrics of reusable building blocks in middleware to detect disruptions. The detection methodology will use model-based analysis to establish baseline behavior in terms of performance metrics at design time. It will then compare the performance metrics extracted from operational data to the baseline performance using well-established statistical techniques in order to flag deviations. A novel aspect of our methodology is the dual-use of performance evaluation and monitoring techniques for the purpose of detection of disruptions.

I. Introduction

Emerging Trends. Society today is increasingly reliant on the services provided by the cyber infrastructure, which requires the correct and continual operation of a multitude of mission-critical applications it hosts. Applications, such as air traffic control, industrial process automation, nuclear reactors, oil refineries, power grids, telecommunication networks, inventory management systems, banking systems and patient monitoring systems enforce stringent performance demands, such as throughput and response time, for their correct operation. These demands must be supported and delivered by the cyber infrastructure, which includes networks, routers and hosts, and services offered by the infrastructure, such as secure transaction services, virtual private networks (VPNs), web portals, and distributed command and control.

A key enabler in realizing the service-oriented cyber infrastructure has been *performance-tuned middleware* [1], as shown in Figure 1. Middleware comprises software layers that provide platform-independent execution semantics and reusable building blocks that coordinate how application components are composed and interoperate. The cyber infrastructure services outlined above are likely to be built using these reusable, patterns-based middleware building blocks.

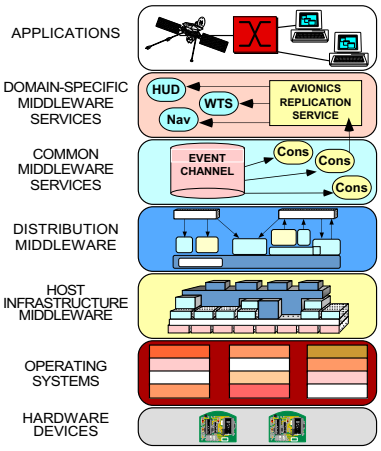

Fig. 1. **Middleware Stack**

Challenges. In the present environment of increased terrorist threats and/or malicious behavior by cyber users, the cyber infrastructure is a prime target for attacks that are aimed at disrupting the services and information flow, thereby bringing down the infrastructure. Additionally, benign failures, such as link or node crashes, or software failures, too may disrupt the normal operation of the services. It is necessary that the cyber infrastructure including the hardware, networks and services be trustworthy to host a wide range of mission critical applications.

Any disruptions could have significant adverse impact on the economy (*e.g.*, disruptions in financial systems, such as stock trading), in some cases on the environment (*e.g.*, disruption in pollutant monitoring systems), or may even result in loss of life (*e.g.*, disruption in 911 emergency services). Ideally, such disruptions must be prevented as much as possible, but no prevention strategy can be completely sufficient. Timely detection, which consists of rapidly identifying the occurrence of the dis-

ISBN 0-7803-9814-9/$10.00 ©2005 IEEE

ruption, then becomes necessary to mitigate the damage caused by the disruption and to restore the services.

Related Work. Prevalent detection strategies are based on three types of data, namely, packet traffic flowing over the network [2], [3], [4], low level system audit data [5], [6] including behavior of routers and link statistics, and log files generated by database servers and web servers [7]. Of these, detection based on the first two types of data streams is significantly mature. Detection based on the lower system layers such as system call traces and network traffic, however, cannot provide adequate protection because the data lacks semantic information.

Solution Approach. A natural remedy to overcome this drawback is to base detection strategies on the data collected at the application layer. However, detection based on the application layer data is invariably specific to the application, and requires custom solutions. Thus, there is a need to develop detection strategies that are based on data collected at higher layers of the infrastructure, which have significant amount of semantic information, but which are not specific to the application. We identify these higher layers to consist of middleware building blocks that are used to build the reusable cyber services, such as VPN. The data that can be collected from these layers of middleware building blocks contain a higher level of semantic information and are yet application-independent. Thus, these provide a rich opportunity for the auditing of these systems.

Since cyber services are *performance-sensitive*, that is, they need to have an acceptable level of performance in order to be useful, we advocate the use of performance metrics of the middleware building blocks and their layered correlation for the purpose of detecting disruptions and mitigating the consequences. Based on the performance metrics, reusable detection techniques which can be uniformly applied in a "plug-and-play" manner across several diverse services offered by the cyber infrastructure will be developed. A significant advantage of our approach is the *dual-use* of performance evaluation and monitoring techniques for the purpose of detection of disruptions.

The rest of the paper is organized as follows: Section II describes our methodology using a VPN service as a case study; and Section III provides concluding remarks and future work.

II. Disruption Detection Approach

This section describes the disruption detection approach which consists of three aspects: (1) establishing baseline behavior in terms of performance metrics using model-based analysis, (2) collecting operational data and extracting the performance metrics from the data, and (3) analyzing the measured performance metrics using well-established statistical techniques to detect disruptions. As a concrete example of a cyber service, we have chosen the virtual router [8] as our case study, which we shall use to describe our methodology.

A. Case Study: Virtual Router

We demonstrate our approach using a case study of an important cyber service, namely, Virtual Private Networks (VPNs). We focus on a crucial artifact of VPN called a virtual router (VR) [8]. We have chosen the virtual router as our case study focus since it exhibits the use of a range of complex design patterns [9] that are codified in reusable middleware building blocks. These traits are the hallmark of any critical cyber service that must be made trustworthy to achieve desired levels of information assurance.

Figure 2 illustrates the architecture of a provider-provisioned virtual private network (PPVPN) [10] using virtual routers (VR). A VR is a software/hardware component that is part of a physical router called the provider edge (PE) router. A VR provides the mechanisms to provide highly scalable, differentiated levels of services in VPN architectures. Multiple VRs can reside on a PE device. VRs can be arranged in a hierarchical fashion within a single PE as shown in Figure 2. Moreover, an entity acting as a service provider for an end customer might itself be a customer of a larger service provider. VRs may also use different backbones to improve reliability or to provide differentiated levels of service to customers.

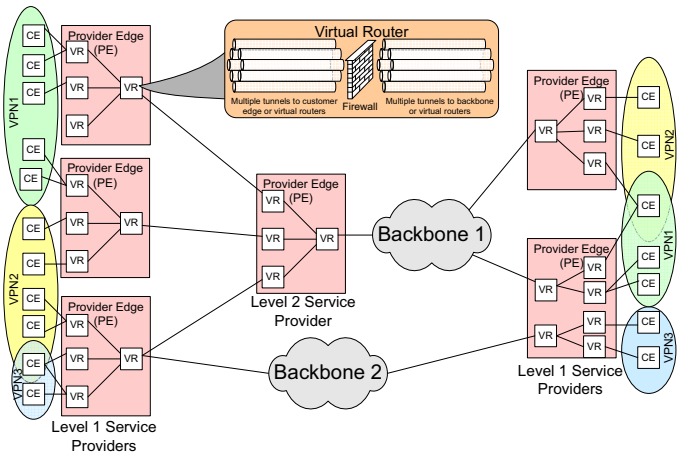

Fig. 2. **VPN Architecture using Virtual Routers**

Customer edge (CE) devices wishing to join a VPN connect to a VR on the PE device. A VR can multiplex several distinct CEs belonging to the same VPN session. A VR may use tunneling mechanisms to use multiple routing protocols and link layer protocols, such as IPSec, GRE, and IP-in-IP, to connect with the CEs. A totally different set of protocols and tunneling mechanisms could be used for inter-VR or VR-backbone communication. These tunneling mechanisms can also be the basis for differentiated levels of service as well as to provide improved reliability. A VR also comprises firewall capabilities.

The VPN-VR architecture with its various overlay network technologies is thus a complex architecture whose trustworthi-

ness is of paramount importance. Implicitly manifested in the VPN-VR architecture are a number of patterns [9] some of which are shown in Figure 3, which are implemented as reusable building blocks. For example, the VR acts as a *Mediator* between a CE and the backbone or other VRs. It also provides an *Adapter* interface between the two since a VR may use totally different communication and routing protocols to connect a CE and to a backbone. The tunneling supported by VR illustrates the *Bridge* and *Facade* patterns. The hierarchy formed by VRs is a form of the *Composite* pattern.

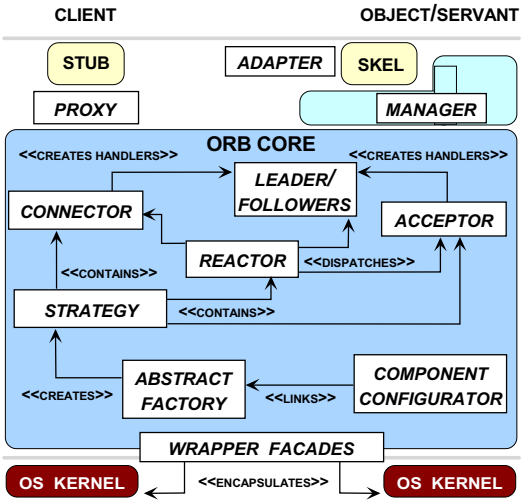

Fig. 3. **Patterns and Pattern Languages in VPN-VR**

Beyond the inherent design patterns [9] exhibited by the VPN-VR architecture, there also exist other complex architectural patterns [11], [12]. For example, the mediating role of the VR can be carried out using the *Broker* pattern, codified by artifacts such as an object request broker [13]. The various event and request (de)multiplexing capabilities in the composite VR model is supported by a form of the *Reactor* pattern. Connection management issues between VRs and CEs and/or backbone devices is managed by the *Connector/Acceptor* patterns, whereas the scalability and concurrency issues arising out of having to support multiple VRs in a PE can be handled using the *Leader/Followers* pattern. Asynchronous request/response and/-or publish/subscribe messaging can be provided by the *Proactor* and *Asynchronous Completion Token* patterns.

This case study illustrates how different patterns can be present in critical cyber services. Our goal therefore is to develop performance models (Section II-B) for the different patterns exhibited by these cyber services. Consequently, we rely on appropriate operational data collection (Section II-C) from all these patterns-based building blocks for analyzing run-time performance of these building blocks and comparing it with expected performance characteristics (Section II-D). Results of such analysis can indicate behavioral anomalies that then can be used to identify potential cyber attacks or general failures.

B. Establishing Baseline Behavior

The first step in the detection methodology consists of developing analytical/simulation performance models for individual building blocks which are used in various services offered by the cyber infrastructure. These models can then be used to determine the performance metrics of each middleware building block for different workloads at design time. These performance metrics can be used to establish baseline behavior of the building blocks.

In order to illustrate how model-based analysis could be used to establish baseline behavior, we consider a VR that provides VPN services to two organizations, with each organization having a customer edge (CE) router connected to the VR. The employees of each organization send VPN set up and tear-down services to the VR via customer edge routers. Also, the VR offers differentiated level of service, with organization #1 receiving prioritized service over organization #2. A *Reactor* pattern could be used to demultiplex the events originating from the two CEs. For simplicity, we consider a single-threaded, *select*-based Reactor implementation.

From the point of view of performance evaluation, the Reactor at the VR has the following characteristics:

- The Reactor accepts two types of input events corresponding to the two different CEs connected to the VR. Each event type has an event handler registered with the Reactor.
- Each event type has a separate queue, which holds the incoming events of that type. The buffer capacity for the queue of type # events is denoted N_1 and of type #2 events is denoted N_2.
- Events of type #1 are serviced with a higher priority over events of type #2. This is necessary to provide differentiated service to organization #1. Thus, in a given snapshot, when event handles corresponding to both event types are enabled, the event corresponding to type #1 is serviced with a priority over event handle of type #2 event.
- Event arrivals for both types of events follow a Poisson distribution with rates λ_1 and λ_2, while the service times of the events are exponentially distributed with rates μ_1 and μ_2.

The following performance metrics are of interest for each one of the event types in the Reactor pattern described above.

- **Throughput** – which provides an estimate of the number of events that can be processed by the single threaded event demultiplexing framework. In the context of the VR, this metric indicates the number of connection set up and tear down requests serviced by the Reactor for each one of the organizations over a period of time.
- **Queue length** – which provides an estimate of the queuing for each of the event handler queues. This metric determines the average waiting time for each event type before it is serviced.
- **Probability of event loss** – which indicates how many events will have to be discarded due to lack of buffer space. This indicates the average probability with which an incoming request has to be dropped.
- **Response time** – which indicates the time taken to service an incoming event. This is the time taken by the VR to fulfill a

request from a CE.

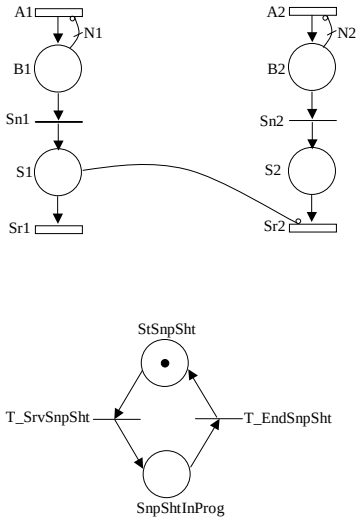

Fig. 4. **SRN Model for the Reactor Pattern**

The performance model of each building block can be developed using well-known modeling paradigms such as Stochastic Reward Nets (SRNs) [14], Markov Regenerative Stochastic Petri Nets (MRSPNs) [15], and Fluid Stochastic Petri Nets (FSPNs) [16]. SRNs, MRSPNs and FSPNs comprise of extensions to Petri nets [17] which have been developed for the purpose of performance analysis. Simulation techniques such as discrete-event [18] and hybrid systems [19] simulation [20] could also be used for performance analysis of building blocks. In this section, we discuss the SRN-based model of the Reactor pattern for the sake of illustration.

Figure 4 shows the SRN model for the Reactor pattern with the characteristics described above. It consists of two parts, namely part (a), which is the top half, and part (b), which is the bottom half of the figure. Part (a) models the arrival, queuing and service of the two types of events. In the figure, transitions $A1$ and $A2$ represent the arrivals of the events of types #1 and #2, respectively. Places $B1$ and $B2$ represent the queue for the two types of events. Transitions $Sn1$ and $Sn2$ are immediate transitions which are enabled when a snapshot is taken. Places $S1$ and $S2$ represent the enabled handles of the two types of events, whereas transitions $Sr1$ and transition $Sr2$ represent the execution of the enabled event handlers of the two types of events. An inhibitor arc from place $B1$ to transition $A1$ with multiplicity $N1$ prevents the firing of transition $A1$ when there are $N1$ tokens in place $B1$. The presence of $N1$ tokens in the place $B1$ indicates that the buffer space to hold the incoming input events of the first type is full, and no additional incoming events can be accepted. The inhibitor arc from place $B2$ to transition $A2$ achieves the same purpose for type #2 events. The inhibitor arc from place $S1$ to transition $Sr2$ prevents the firing of transition $Sr2$ when there is a token in place $S1$. This models the prioritized service for the events of type of #1 over events of type #2.

Part (b) of the net models the process of taking successive snapshots and prioritized service of event handles corresponding to type #1 event in each snapshot as explained below. Transition $Sn1$ is enabled when there is a token in place $StSnpSht$, at least one token in place $B1$ and no tokens in place $S1$. Similarly, transition $Sn2$ is enabled when there is a token in place $StSnpSht$, at least one token in place $B2$ and no tokens in place $S2$. Transition $T\_SrvSnpSht$ is enabled when there is a token in either one of the places $S1$ and $S2$, and the firing of this transition deposits a token in place $SnpShtInProg$.

The presence of a token in the place $SnpShtInProg$ indicates that the event handles that were enabled in the current snapshot are being serviced. Once these event handles complete execution, the current snapshot is complete and it is time to take another snapshot. This is accomplished by enabling the transition $T\_EndSnpSht$. Transition $T\_EndSnpSht$ is enabled when there are no tokens in both place $S1$ and $S2$. Firing of the transition $T\_EndSnpSht$ deposits a token in place $StSnpSht$, indicating that the service of the enabled handles in the present snapshot is complete which marks the initiation of the next snapshot. Table I summarizes the enabling/guard functions for the transitions in the net.

The performance model of a building block can be solved numerically using techniques incorporated in well-known tools such as SPNP [21], or using simulation [18] to obtain quantitative estimates of the performance metrics for different values of the input parameters. Next, we describe how model-based analysis can be used to establish the baseline behavior of a building block. The baseline behavior of a building block is defined in terms of the expected or the average values of the performance metrics. The average values of these metrics will be obtained using the following steps:

- Identify the service profile, which consists of modes of operation of the service which uses the building block.
- Estimate the likelihoods or the occurrence probabilities of each mode of operation.
- Estimate the values of the input parameters for each mode of operation.
- Solve the performance model of the building block using the input parameter values to estimate the values of the performance metrics for each mode of operation.
- Obtain the expected estimates of the performance metrics based on the performance metrics and the occurrence probabilities of each mode.

Once again, we explain the above steps with the help of the Reactor pattern. Consider a scenario where the VPN service provided to one of the organizations connected to the VR is used by the organization to provide remote access to its employees. The employees use this service to set up and tear down secure connections to the organization's infrastructure. The organization has grouped its employees into two categories, namely, technical and administrative. The technical employees send

TABLE I
Guard functions

Transition	Guard function
$Sn1$	$((\#StSnpShot == 1)\&\&(\#B1 >= 1)\&\&(\#S1 == 0))?1 : 0$
$Sn2$	$((\#StSnpShot == 1)\&\&(\#B2 >= 1)\&\&(\#S2 == 0))?1 : 0$
$T\_SrvSnpSht$	$((\#S1 == 1)\|\|(\#S2 == 1))?1 : 0$
$T\_EndSnpSht$	$((\#S1 == 0\&\&(\#S2 == 0))?1 : 0$

TABLE II
Performance metrics per mode of operation of VPN

Performance metric	Normal mode	Inclement mode	Average
T_1	0.40/sec	0.91/sec.	0.4510/sec.
T_2	0.40/sec	0.91/sec.	0.4510/sec.
Q_1	0.12	1.86	0.2940
Q_2	0.12	1.86	0.2940
L_1	0.09	0.21	0.0291
L_2	0.09	0.21	0.0291

connection set up requests to the VPN at rate λ_1, and the administrative employees send connection set up requests to the VPN at rate λ_2. The VPN service has two modes of operation, normal and inclement. In the normal mode, on a daily basis some of the employees of the organization have negotiated telecommute plans and use the VPN. In the inclement mode, on a given day when bad weather creates hazardous driving conditions a larger number of employees choose to telecommute leading to higher values of the rates of set up requests λ_1 and λ_2, than what are observed in the normal mode of operation. We assume that $\lambda_1 = 0.5/sec$ and $\lambda_2 = 0.5/sec$ in the normal mode of operation. In the inclement mode of operation, the values of $\lambda_1 = 1.0/sec$ and $\lambda_2 = 1.0/sec$. The values of the service rates μ_1 and μ_2 are the same in both the modes of operation. Also, the buffer capacities of both types of events N_1 and N_2 are the same in both modes. For the sake of illustration, we assume that the $\mu_1 = \mu_2 = 2.0/sec$, and $N_1 = N_2 = 5$. Using the SRN model we compute the values of the performance metrics for these two modes of operation, and these values are summarized in Table II. Further, we assume that on 90% of the days the VPN service operates in the normal mode, and only on 10% of the days the service operates in the inclement mode. The average values of the performance metrics are computed as the weighted sum of the performance metrics for each mode, with the weights given by the probabilities of occurrence for each mode. The average performance metrics are also reported in Table II. The above process will also create a probability distribution for each one of the performance metrics.

In the above example, for the sake of illustration, modes of operation were defined on a per day basis. It is conceivable that the modes of operation are defined at a finer level of granularity. For example, a day could be divided into three intervals and each interval could have two modes of operation. In this case, an estimate of the average performance metrics could be obtained for each one of the intervals using the method described above. In general, the finer the granularity of defining the baseline behavior, the better is the detection capability. However, the improved detection capability comes at a higher cost of data collection and analysis.

C. Collecting Run-time Operational Data

This section describes the second facet of our detection methodology comprising collecting run-time operational data from the different middleware building blocks and using this data to extract the performance metrics of the building blocks.

Our methodology relies on the appropriate configuration of different layers of the distributed system for operational data collection. The audit logs collected at various layers can then be mined, filtered and fused to decipher and detect any service disruptions. The concept of logging is not new. For example, protocols like SNMP [22] require logging of data in MIBs. However, in today's state of the art, logs collected at lower layers, such as that at the routers, are agnostic about the applications and their behavior. Therefore, the algorithms that operate on the MIB data cannot distinguish application traits in the collected data. At the same time, applications cannot make sense of all the logging that routers provide nor can they access, in many instances, all the logging that lower layers of the system provide.

Our detection methodology will use reusable, multi-layered, middleware logging service capabilities that can be appropriately configured to collect and fuse operational data at multiple layers of the cyber infrastructure. Our assumption is that contemporary middleware provide appropriate capabilities that enable external services to configure appropriate types and amount of operational data collection strategies within the multiple layers of the middleware. There are many approaches to retrofit existing middleware with these capabilities, such as using interceptors [23] or aspect weaving technology [24] that could be used to enhance existing middleware to provide such capabilities. Below we provide a sampling of the kinds of operational data that need to collected. We use the case study of the virtual router described in Section II-A and the reactor pattern model described in Section II-B in providing the list.

- **Number of event handlers** – this data indicates the number of independent input events that a reactor must demultiplex and handle. For the VR, this corresponds to the number of incoming CE connections into a VR and/or number of incoming VR

connections into a second level VR. This data could be further divided into finer level details, such as the number of tunnels supported per link.

- **Priority of event handlers** – this data corresponds to the priority at which the reactor must handle an incoming event. For the VR, this may correspond to the differentiated levels of service offered to the customers.
- **Throughput** – this data corresponds to the number of events for all tunnels in a channels served, number of events per channel, and number of events for all channels together served. This metric is a good indicator of whether differentiated levels of service are being offered or not.
- **Queing delays and sizes** – this data corresponds to the number of events waiting in the various queues to be serviced. For the VR, this metric indicates the number of incoming requests queued up waiting to be serviced. The waiting times in these queues directly affects the response time.
- **Event losses** – this data corresponds to the number of incoming events that could not be handled due to lack of resources. In the case of VR, this data corresponds to the number of incoming requests that could not be satisfied due to lack of available resources. This data is also a good metric to determine how differentiated services are handled.
- **Number of physical communication links and link statistics** – this data corresponds to the actual number of physical links connected to a PE and the number of VR channels multiplexed over it. Additionally, the metrics of interest are per link congestion and flow control statistics. These statistics have an effect on the workload characteristics of the reactor model.
- **Node-specific metrics** – this operational data corresponds to the node that acts as a PE. This data includes current CPU load, number of threads created and in use by the VR hierarchy.

A set of similar operational data can be defined for each one of the patterns exhibited by cyber services.Instrumentation necessary for the purpose of data collection could adversely impact the performance. To mitigate the instrumentation overhead, adaptive configuration of the amount of data and the level of detail may be necessary. Also, the data collection outlined above may incur significant levels of accidental complexities when these adaptations and configurations are defined in *ad hoc* ways, such as manually figuring out the right levels of data to be collected. To alleviate these complexities, model-driven generative tehnologies to configure the appropriate levels of data collection at different layers of the cyber infrastructure may be used. The adaptation strategies for data collection will be based on model-based synthesis of predictive control for adaptive data collection services.

The operational data collected in this fashion can be filtered and distributed to central facilities where performance analysis and disruption detection will be carried out. Existing protocols such as syslog [25] may be enhanced and used, in conjunction with middleware provided publisher-subscriber services for reliable delivery of operational data to the central facilities. This data is then collected in central databases similar to MIBs. The goal is then to extract the performance metrics of the building blocks including throughput, response time and event loss probability.

D. Analyzing Performance Data for Disruption Detection

This section describes the third facet of our detection methodology comprising statistical analysis of performance data for detecting disruptions.

In each analysis window, for every performance metric extracted from the log data we will compute the exponential moving average [26] to summarize the metrics generated in the past several windows. The approximate weight of each window is determined by the smoothing constant used to compute the exponential moving average. Using the exponential moving average of each performance metric, we will compute a probability score using the χ^2 test [26]. The computation of the probability score will be based on the expected value of the metric estimated at design time. The anomaly score will determine the degree of departure of the service based on the performance metric. Bayesian networks [27] may be used to correlate the anomaly score based on each performance metric to obtain the overall anomaly score for the entire building block. For example, in the case of the Reactor pattern, an anomaly score will be generated based on each one of the performance metrics, namely, throughput, queue length, and loss probability for each one of the event types, and an overall anomaly score correlating the scores based on each one of the performance metrics.

In order to classify the behavior of a service as anomalous, we will correlate the anomaly scores of the different building blocks used to implement the service. These building blocks could reside at multiple layers. We explain this with the help of an example of the VPN service provided by the VR used in Section II-B. In addition to the Reactor pattern used at the VR to demultiplex the events arriving from two CEs, a Reactor pattern could also be used at each CE to multiplex the events arriving from each one of the two types of employees. An anomaly score will also be obtained for the Reactors used in each one of the CEs. To determine if the VPN service of organization #1 is disrupted, anomaly scores between the Reactor in the CE of #1 and the Reactor at the VR will be correlated using a Bayesian network. Similarly, anomaly scores of the Reactor in CE #2 and VR will be correlated to determine if the VPN service of organization #2 is disrupted.

Correlation of anomaly scores is used in a hierarchical manner to reduce the false positive rate that is common in anomaly detection [27]. Initially, correlation between the anomaly scores obtained from the analysis of each performance metric is performed conducted to obtain the overall anomaly score for a building block. Subsequently, the anomaly scores of the different building block are correlated to obtain the overall anomaly score for the entire service. In addition to reducing the false positive rate, correlation may also enable us to isolate the source of the disruption. For example, consider the situation where the VPN service of the organization #2 is under attack. If detection

were to be based solely on the score of the Reactor pattern at the VR, then it would indicate that the behavior of both the VPN services are anomalous. However, by correlating the scores at multiple layers, the source of the disruption, namely, organization #2 may be identified. Identification of the source may also suggest strategies to contain the disruption and to restore the services.

III. Concluding Remarks and Future Research

Due to the increasing reliance of our day to day lives on the services provided by the cyber infrastructure, it is necessary to detect the disruptions to the infrastructure in a timely manner. In this paper we describe a disruption detection methodology that uses the performance metrics of the middleware building blocks for the purpose of detection. The methodology consists of the following steps: (1) establishing at design-time, the baseline behavior defined in terms of the performance metrics of the middleware building block(s) used to implement the services using model-based analysis, (2) logging of operational data and extracting performance metrics of the building block(s) from the operational data during service operation, and (3) comparing the run-time performance metrics with the expected baseline behavior to flag departures from the norm. Our methodology uses semantically rich data collected at the higher layers of the infrastructure, that is sufficiently generic across a multitude of services. A novel aspect of our methodology is the *dual-use* of performance evaluation and monitoring techniques for the purpose of detection of disruptions.

Our future work includes evaluating our techniques in the Emulab [28] testbed. Evaluating the trustworthiness of the overall cyber infrastructure will require determining the effectiveness and timeliness of the disruption detection techniques, which compare the collected run-time performance statistics to design-time estimates of performance. Disruptions within the system will be triggered by introducing background traffic that could mimic a malicious attack. Similarly, benign disruptions, such as failure of links or nodes will also be emulated. The detection performance of our techniques will be evaluated using ROC-like curves used in signal detection [29], [30].

References

[1] R. E. Schantz and D. C. Schmidt, "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," in *Encyclopedia of Software Engineering* (J. Marciniak and G. Telecki, eds.), New York: Wiley & Sons, 2002.

[2] H. G. Kayacik and A. Zincir-Heywood, ""A case study of three open source security management tools"," in *IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 03)*, 2003.

[3] J. Stevens and S. Saniepour, ""SecureDirect: Proactive security through content–based traffic control"," in *Proc. of the 17th Intl. Conference on Advanced Information Networking and Applications (AINA 03)*, 2003.

[4] A. H. Sung and S. Mukkamala, ""Identifying important features for intrusion detection using support vector machines and neural networks"," in *Proc. of the 2003 Symposium on Applications and the Internet (SAINT 03)*, 2003.

[5] R. Chinchani, S. Upadhyaya, and K. Kwait, ""Towards the scalable implementation of a user level anomaly detection system"," in *Proc. of MILCOM*, 2002.

[6] L. Zhuowei, A. Das, and S. Nandi, ""Utilizing statistical characteristics of N–grams for intrusion detection"," in *Proc. of 2003 Intl. Conference on Cyberworlds*, 2003.

[7] T. Ryutov, C. Neuman, D. Kim, and L. Zhou, ""Integrated access control and intrusion detection for web servers"," in *Proc. of the 23rd Intl. Conference on Distributed Computing Systems (ICDCS 03)*, 2003.

[8] P. Knight, H. Ould-Brahim, and B. Gleeson, "Network based VPN Architecture using Virtual Routers," *IETF Network Working Group Internet Draft, draft-ietf-l3vpn-vpn-vr-02.txt*, pp. 1–21, Apr. 2004.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[10] A. N. (editor), "Generic Requirements for Provider Provisioned Virtual Private Network (PPVPN)," *IETF Network Working Group Request for Comments, RFC 3809*, pp. 1–25, June 2004.

[11] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture—A System of Patterns*. New York: Wiley & Sons, 1996.

[12] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. New York: Wiley & Sons, 2000.

[13] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 3.0.2 ed., Dec. 2002.

[14] A. Puliafito, M. Telek, and K. S. Trivedi, "The evolution of stochastic Petri nets," in *Proc. of World Congress on Systems Simulation*, (Singapore), pp. 3–15, September 1997.

[15] H. Choi, V. Kulkarni, and K. S. Trivedi, "Markov Regenerative Stochastic Petri Net," *Performance Evaluation*, vol. 20, no. 1–3, pp. 337–357, 1994.

[16] G. Horton, V. Kulkarni, D. Nicol, and K. S. Trivedi, "Fluid stochastic Petri nets: Theory, application and solution techniques," *Journal of Operations Research*, vol. 405, 1998.

[17] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.

[18] The VINT Project, "Network Simulator - NS-2." `http://www.isi.edu/nsnam/ns`.

[19] T. A. Henzinger and S. Sastry, eds., *Hybrid Systems: Computation and Control - Lecture Notes in Computer Science*. New York, NY: Springer Verlag, 1998.

[20] S. Bohacek, J. Hespanha, J. Lee, and K. Obraczka, "A hybrid systems modeling framework for fast and accurate simulation of data communication networks," in *Proceedings of ACM SIGMETRICS '03*, June 2003.

[21] C. Hirel, B. Tuffin, and K. S. Trivedi, "SPNP: Stochastic Petri Nets. Version 6.0," *Lecture Notes in Computer Science 1786*, 2000.

[22] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)," *IETF Network Working Group Request for Comments, RFC 1157*, pp. 1–35, May 1990.

[23] D. C. Schmidt and S. Vinoski, "CORBA Metaprogramming Mechanisms, Part 1: Portable Interceptors Concepts and Components," *C/C++ Users Journal*, Mar. 2003.

[24] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-Oriented Programming," in *Proceedings of the 11th European Conference on Object-Oriented Programming*, pp. 220–242, June 1997.

[25] C. L. (editor), "The BSD Syslog Protocol," *IETF Network Working Group Request for Comments, RFC 3164*, pp. 1–29, Aug. 2001.

[26] R. A. Johnson and D. W. Wichern, *Applied Multivariate Statistical Analysis*. Upper Saddle River, NJ: Prentice Hall, 1998.

[27] Y. Wu, B. Foo, Y. Mei, and S. Bagchi, ""Collaborative intrusion detection system (cids): A framework for accurate and efficient IDS"," in *Proc. of 19th Annual Computer Security Applications Conference (ACSAC 2003)*, 2003.

[28] Robert Ricci and Chris Alfred and Jay Lepreau, "A Solver for the Network Testbed Mapping Problem," *SIGCOMM Computer Communications Review*, vol. 33, pp. 30–44, Apr. 2003.

[29] J. Hancock and P. Wintz, *Signal Detection Theory*. New York: McGraw–Hill, 1966.

[30] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman, ""Evaluating intrusion detection systems: The 1998 DARPA off–line intrusion detection evaluation"," in *Proc. of DISCEX 2000*, 1999.