# Remote Management of Boundary Protection Devices with Information Restrictions

**Aaron Adler**
BBN Technologies
10 Moulton Street
Cambridge, MA 02138
aaronadler@alum.mit.edu

**Peter Samouelian**
BBN Technologies
10 Moulton Street
Cambridge, MA 02138
psamouelian@acm.org

**Michael Atighetchi**
BBN Technologies
10 Moulton Street
Cambridge, MA 02138
michael.atighetchi@raytheon.com

**Yat Fu**
Air Force Research Lab
26 Electronic Parkway
Rome, New York 13441
yat.fu@us.af.mil

## Abstract

Boundary Protection Devices (BPDs) are used by US Government mission partners to regulate the flow of information across networks of differing security levels. BPDs provide several critical functions, including preventing unauthorized sharing, sanitizing information, and preventing cyber attacks. Their application in national security and critical infrastructure environments (e.g., military missions, nuclear power plants, clean water distribution systems) calls for a comprehensive load monitoring system that provides resilience and scalability, as well as an automated and vendor neutral configuration management system that can efficiently respond to security threats at machine speed. Their design as one-way traffic control systems, however, presents challenges for dynamic load adaptation techniques that require access to application server performance metrics across network boundaries. Moreover, the structured review and approval process that regulates their configuration and use presents two significant challenges: (1) Adaptation techniques that alter the configuration of BPDs must be predictable, understandable, and pre-approved by administrators, and (2) Software can be installed on BPDs only after completing a stringent accreditation process. These challenges often lead to manual configuration management practices, which are inefficient or ineffective in many cases. The Hammerhead prototype, developed as part of the SHARC project, addresses these challenges using knowledge representation, a rule-oriented adaptation bundle format, and an extensible, open-source constraint solver.

DISTRIBUTION A: Approved for public release; distribution unlimited (Case Number 88ABW-2017-4436).
Work sponsored by AFRL under contract FA8750-16-C-0056; the views and conclusions contained in this document are those of the authors and not AFRL or the U.S. Government.

## 1 Introduction

BPDs are similar to gateways or firewalls except for key differences that improve their security and the security of the networks they bridge. For example, the types of information they are configured to transfer across networks is preconfigured and heavily scrutinized according to mission requirements. Multi-step and redundant filters perform deep content inspection, analyzing data across multiple OSI layers and packets to ensure that the intent of the data transfer is in compliance with the BPD's transfer policies.



Another key difference is that some BPDs are designed at the hardware level to prevent two-way traffic by using dedicated circuits for their sending and receiving network interfaces, which are in turn separated by one-way data diodes. This hardware architecture protects the receiving network from data exfiltration attacks. The one-way data diodes also prevent software failures and high load conditions for critical functions such as data filtering, auditing and logging that occur on the BPD's receiving side to signal the sending side. This is by design as it limits an adversary's ability to learn the effectiveness of various attack strategies and, hence, the target's security posture.

Modern information systems include load balancers (LBs) and application clustering to achieve resiliency, high availability, scalability, and security. In a web-based application, for example, LBs provide these benefits by ensuring Internet-based requests are routed to available servers across an organization's server pool. Hence, in their role as traffic managers between network boundaries, LBs require server statistics on the receiving network in order to make good routing decisions. This information includes each server's basic online status, CPU utilization, available bandwidth, request completion rate, and other metrics. BPDs are designed to prevent all such types of feedback (including the success or failure of data transfers) to the sending network from their receiving side.

In practice, some BPDs are equipped with pairs of one-way data diodes to support separate directional data flows, as shown in Figure 1 between an open / low security network and a restricted / enclave / higher security network, in this paper referred to as 'open' and 'restricted.' Sending and receiving interfaces may also reside on completely separate BPDs, potentially across vendor types. Communication techniques that can leverage these separate directional dataflows in a secure and coordinated manner would allow deployments to better leverage commodity LBs to improve scalability and fault tolerance of BPD deployments.

BPD configuration and administration tasks generally require physical access to the BPD and potentially multiple administrators working separately on each network interface. While this process provides transparency and effective countermeasures against some types of security threats (e.g., insider threats), it is ineffective or inefficient when responding to cyber attacks, which operate at machine speed, or sup-

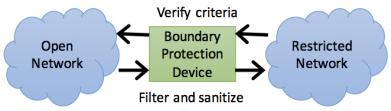


Figure 1: The BPD connects open and restricted networks.

porting ongoing maintenance, such as updating virus definitions across multiple BPDs. A vendor-neutral configuration management solution that satisfies the unique constraints of BPDs while leveraging automation would lower the maintenance costs of BPDs as well as improve their security.

A system that provides a comprehensive approach that efficiently monitors, configures, and allocates BPDs using transparent and reviewable techniques would significantly improve the state-of-the-art in their administration and deployment. The Hammerhead prototype, developed on the SHARC project, addresses these challenges using a combination of AI components, including: 1) Device and behavior abstractions (Knowledge Engineering), 2) A rule-oriented, XML-based adaptation bundle format, and 3) A constraint solver and planner. These components are combined with a core Observe-Orient-Decide-Act (OODA) loop architecture and a publish/subscribe messaging bus. The following sections show how the unique requirements of BPDs are addressed by these components, beginning with a system overview.

## 2 System Overview

Figure 2 shows an overview of a potential Hammerhead deployment. A Hammerhead instance on the open network receives traffic sensor input on data flows entering the BPD pool and sends aggregated status information to the Hammerhead instance on the restricted network, which is the administration or owning instance of the BPD pool. The aggregated information conforms to pre-approved XML data that, like all data flows, is procesed through a BPD. This information is key to enabling the reasoning module in the restricted-side Hammerhead instance, which compares the received status information to information available on the restricted side of the BPD pool to detect potential choked data flow conditions or other anomalies.

The restricted Hammerhead instance may take actions to address anomalies, including migrating software across BPD instances or bringing standby instances online to handle traffic spikes. The use of replica sets (multiple BPDs with identical policies) facilitates the system's ability to scale or failover as needed. A policy in this context is the software and configuration needed to process a type of data flow (e.g., policies exist for video streaming data flows and web service data flows). Actions that require configuration changes to the LBs on the open side are routed to the open-side Hammerhead instance via a BPD as pre-approved XML messages. The use of coarse-grained, pre-approved XML messages to transmit open-side traffic conditions and LB configuration changes provides an opportunity to securely apply dynamic load balancing techniques without compromising the policies imposed by the BPDs. Low-level details

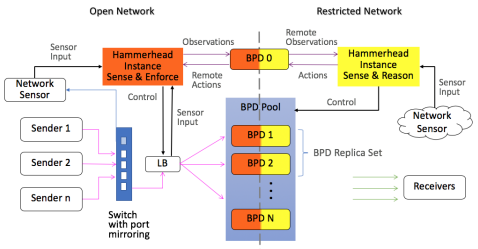


Figure 2: Hammerhead system overview

on data flows entering the BPD and BPD health-metrics are not revealed outside their respective networks. LB routing is configured using control messages from the restricted Hammerhead instance.

The deployment's topology (i.e., which BPDs are being monitored, which LBs are in use, and the replica set configuration), the specific rules that govern the reasoning process on the restricted Hammerhead instance, and all the defined sensors and actuators are described in a rule-oriented XML-based adaptation bundle (see Section 6). This format significantly improves the administrators' ability to review and adopt effective and dynamic adaptation techniques in mission critical BPD deployments.

Moreover, the solution avoids the installation of new software on the BPD and the associated re-accreditation process. BPD configuration changes can be executed on the BPD using either the Guard Remote Management Protocol (GRMP), an emerging, vendor-neutral standard that allows for secure remote BPD administration or, alternatively, through custom actuators that use the BPD's native-management interface. As described later, BPD status information (like CPU utilization) can be accessed using a BPD SNMP (Simple Network Management Protocol) MIB (Management Information Base). This provides a greater level of information useful for finding optimal assignments of policies to BPDs.

## 3 Architecture

A core aspect of Hammerhead's architecture is an OODA loop which processes sensor information from a variety of sources, orients and decides on a sequence of actions to take, and then uses a set of actuators to modify the LB or BPD configuration. The orient and decide portions of this loop are implemented using several AI techniques which are described later. The Observe component of the OODA loop is driven by an extensible sensor-based architecture that includes a packet sensor on the network. In order to measure network traffic, we use TShark (www.wireshark.org), a terminal based packet analyzer. The resulting log data produced by TShark is processed by Logstash (www.logstash.net) and ingested into an Elasticsearch database (www.elastic.co).

Each BPD processes one or more flows. Each flow is a single route for a specific type of data that is allowed through the BPD. Hammerhead periodically queries Elasticsearch for the number of network packets sent to and from each

BPD flow. Since TShark supports extensive packet filtering capabilities, a dedicated packet monitor can be created for every BPD flow in a system.

Data from these sensors is fed into a local publish/subscribe message bus as fine-grained events (e.g., the number of HTTP requests sent to a BPD flow within an interval of time). Multiple subscribers on the bus perform dedicated tasks. One such subscriber is responsible for inserting each event into the fact base of a rule engine session (www.drools.org). One important function the rule engine serves is event aggregation. Real-time traffic monitoring solutions can generate high volumes of data that must be filtered and summarized (aggregated) in order to focus on the salient aspects of traffic conditions. Hammerhead filters data at both the TShark and rule engine level; the latter, however, is responsible for aggregating fine-grained events on the status of individual data flows entering or exiting the BPD device into coarse-grained events on the status of a replica set. When Hammerhead observes, for example, that some inbound flows to a replica set are receiving data yet none of the replica set's outbound flows are sending data, it creates a new observation indicating a potential choked traffic condition on the cluster. This can be the result of a bad filter configuration across the cluster or signs of a cyber attack. Event aggregation creates more readable rules, which in turn facilitates the BPD review process and ultimately the trust administrators place on the system.

A second subscriber on the message bus monitors high-level observations to dispatch messages to the Hammerhead instance responsible for the message recipient (a deployment will likely consist of more than two domains so messages must be routed appropriately). Messages are dispatched by comparing their target (i.e., which BPD or LB) with a topology of the system contained in the adaptation bundle.

Actions are executed by actuators in Hammerhead. An actuator exists for the GRMP protocol and is responsible for migrating policies across BPDs, starting and stopping BPDs, and performing other administrative tasks. Since the prototype uses a DNS server as a load balancer, Hammerhead includes an actuator for adding and removing DNS type A and service records. This is used for implementing BPD failover and basic DNS round-robin load balancing. Similar to observations, actions, triggered by rules on the reasoning Hammerhead instance, are placed on the local message bus. A dedicated subscriber monitors actions on the bus and determines how to route the action to a Hammerhead instance based on the topology in the adaptation bundle.

## 4 Overview of AI Techniques

The following sections describe Hammerhead's knowledge engineering, adaptation bundle format, and constraint solver and planner solution. Collectively these techniques model the system and identify faults and optimizations while meeting the requirement for human understandability and review.

## 5 Knowledge Engineering

Hammerhead is designed to work in diverse deployment environments that may contain a variety of BPDs, LBs, and sensor capabilities. Data flows may use one or more BPDs depending on the volume of data and the computational intensity of the required processing. We created a core set of rules that, once understood by administrators and accreditors, are applicable to different situations. We achieved this by using knowledge engineering to develop abstractions that allow the same high level behavior to occur while being agnostic to the source of the sensor data. For example, some BPDs might support the SNMP protocol, others the GRMP protocol, and others may require a custom protocol. By modeling the key characteristics of these devices, the high level rules can be agnostic to the source of the sensor data.

We constructed a hierarchical model (Figure 3) that abstracts away details so that administrators can reason about devices at a higher level. The lowest level of the model is the raw sensor data on replicas (copies) of a particular flow. The next level maps the raw information into observations about flow instances. Next, the state of a policy on a replica is captured, and the following layer determines the state of a replica set. For example, a replica set may have data flowing on some replicas but not others. At the top level, the observations are turned into a ticket that represents a problem to be solved. High level modeling occurs only in the restricted network.

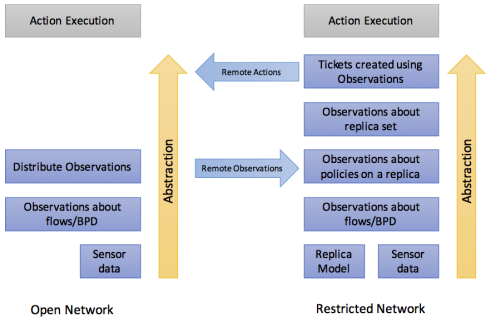


Figure 3: Overview of the rule abstraction layers in each network.

Hammerhead rules are designed to build upon each other and form a model of the system state that can be analyzed. For example, a BPD typically has input and output queues that surround the actual filtering and processing of data. The queue depth increases if the BPD cannot keep up with the rate of incoming data or if the BPD encounters errors. For example, the BPD is designed to fail closed and will not process more data if it cannot write logs to disk (i.e., the disk is full). An increased queue depth could be ascertained directly from the BPD using SNMP or indirectly by observing a change in network flows (data going out of the BPD at a reduced rate relative to an average across time or to the input rate). The full state of the system across network boundaries must be factored into the analysis.

Hammerhead uses tickets to model both the issue and resolution steps. A ticket can be created to alert an administrator, rebalance flows for increased load, or route traffic around a malfunctioning BPD.

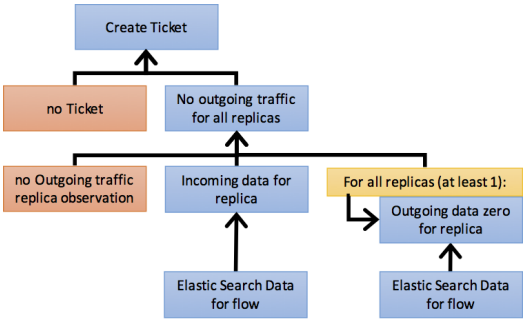


Figure 4: Example of a ticket creation rule chain.

```
<adaptationbundle xmlns="http://arc.bbn.com/AdaptationBundle">
   <id>9f9772c9-5546-40a6-968c-8b7e6926cd44</id>
   <version>1</version>
   <services>
   <triggers>
   <actions>
   <rules>
   <replicas>
       <replicaSet>
   </replicas>
   <cell>
</adaptationbundle>
```

Figure 5: Summary of Adaptation Bundle structure.

Figure 4 shows the observations that culminate in ticket creation for a set of flows that have incoming data but no outgoing data. When a ticket includes an action to reroute traffic, the constraint solver (see Section 7) is used to reassign policies to available BPDs.

## 6 Adaptation Bundle

To satisfy the requirements for human review and approval, Hammerhead encapsulates all adaptation rules, actuator and sensor configuration, and the BPD/LB topology in a single XML file or "bundle." The bundle format provides a complete and explicit enumeration of how the system will behave. This empowers administrators to review each system rule in the context of a specific deployment. Each bundle must be signed by an approving authority before it can be used. A summary of the bundle structure is shown in Figure 5. The largest section contains the declarative rules (using the Drools native rule language) that trigger ticket actions. The replicas and cell sections describe the BPD policy configuration and Hammerhead instances. Actuators and sensors are described in the services section.

A key advantage of the bundle format is that it places restrictions on Hammerhead's input sources (sensors), range of behavior (rules), and scope (BPD topology). The scope of an administrator's approval extends precisely to these components. Together with the bundle's human-readable XML format, this increased approval scope satisfies understandability and review requirements.

## 7 Constraint Solver

Mission planners rely upon BPDs for resource intensive tasks such as full motion video streaming, stateful deep content inspection, and other complex data filtering. To satisfy mission objectives, SHARC must determine an optimal assignment of the policies that filter data flows to hardware resources. Planning must take place at both pre- and in-mission phases of operation. Optimal states are functions of multiple factors, including hardware specifications and accreditation status, mission and policy requirements, and real time resource utilization metrics, such as CPU load and free memory. An optimal assignment must satisfy multiple objectives, including: (1) Hardware constraints such as memory, CPU speed, and bandwidth required by a mission's policies and (2) Cluster-level requirements, such as concurrency and fault tolerance. In addition to the Optaplanner constraint-satisfaction engine (www.optaplanner.org), the scoring function and rule-based planner are the key components used to address these constraints.

### 7.1 Scoring Function

The default scoring function computes a hard and soft score for each candidate solution. Hard scores are evaluated for scalar and binary constraints on most resources, whereas soft scoring is applied to reward cluster-level resiliency characteristics. A hard and soft score of zero indicate that the constraints are satisfied, while negative scores indicate that the constraints are unsatisfied. Hard constraints are used to identify and discard infeasible solutions, whereas soft constraints rank feasible solutions.

**Scalar constraints on consumable hardware resources** These include, for example, policy, memory, and bandwidth requirements. Specifically, the total hard score assignment for consumable resources is:

$$\sum_{d}^{|bpd|}\sum_{r}^{|R|}\begin{cases} C_{(r,d)} - R_{(r,d)} & R_{(r,d)} > C_{(r,d)} \\ 0 & R_{(r,d)} \leq C_{(r,d)} \end{cases}$$

where $|bpd|$ is the number of boundary protection devices, $|R|$ is the number of consumable resources, and $C_{(r,d)}$ is the capacity for resource $r$ on device $d$. $R_{r,d}$ is simply the sum of the values for resource $r$ on all policies assigned to device $d$ in the current solution:

$$R_{r,d} = \sum_{p}^{|P_d|} p_r$$

where $|P_d|$ is the number of policies assigned to device $d$, and $p_r$ is the policy's requirement for resource $r$.

**Scalar constraints on non-consumable resources** Examples are CPU clock speed and number of cores. Such hardware resources are viewed, for the purposes of simplification, as resources that are available equally to all policies assigned to the device; i.e., a policy's CPU speed requirement does not reduce the available CPU speed for other policies. Though this is a simplification, other runtime factors, like CPU load factor, provide a more accurate measure of the cumulative impact policies have on resources that are mapped as non-consumable.

**Discrete resource constraints** Examples include a policy's requirements for isolation or a device's broad capabilities (e.g., to support a type of content). These are implemented as the binary hard score constraints $(0, -1)$.

**Cluster-level constraints** These constraints are designed to reward high-level resiliency characteristics, including even workload balancing and concurrency. In order to create policy deployments that are fault tolerant and capable of satisfying target workloads, the scoring function must reward configurations that evenly distribute load and replication at both the pre-mission planning stage and at runtime. Pre-mission assignments consider a policy's expected load on a BPD based on its memory, bandwidth, and CPU requirements using a simple multiplicative model:

$$p_{load} = \prod_{f}^{|F|} p_f^{W_f}$$

Each policy feature $p_f$ is weighted appropriately using $W_f$ under experimentation. The multiplicative model does not require feature scaling; however, in practice, it is convenient to apply a min-max normalization over a restricted integer range to avoid data overflow errors, which easily occur even for 64-bit integers when using a mulitplicative scoring function. Hammerhead uses a min-max normalization function that projects each feature's zero-based, absolute range to $0 - 100$. For example, processor speed is mapped from $0 - 5000$ Mhz to $0 - 100$ (the source range's minimum and maximum values are not derived from the current observation set). This preserves proportionality and data-independence (e.g., the optimizer does not reorder policy workload assessments as new policies are added or existing policies are removed). Moreover, all values are converted to integers to avoid floating-point arithmetic errors.

The scoring function computes the workload soft score for each candidate assignment of policies to BPDs presented by the heuristic-search engine as follows:

$$soft_{load} = -\big[\sum_{d}^{|bpd|}\sum_{p}^{|P_d|}((p_{load} - load_{avg})^2)\big]$$

where $load_{avg}$ is computed as:

$$load_{avg} = \frac{1}{|bpd|}\sum_{d}^{|bpd|}\sum_{p}^{|P_d|} p_{load}$$

At runtime, the scoring function switches to using runtime factors reported by the SNMP agent, such as load factor and free memory. (Hammerhead currently uses load factor.)

Scoring for reliability and concurrency are handled using similar non-linear models as shown above. Concurrency may be expressed at a number of granularity levels, from ethernet packets to application level messages or active sessions. Excessive reliability and concurrency is penalized using a squared variance as shown above for $soft_{load}$, rewarding solutions which are efficient (i.e., those that minimize the level of resources used to achieve a target level for a constraint). One problem, however, with the above approaches is that the constraint formulas create symmetric regions of penalty and reward around their respective targets, which fails to capture the concept of minimum target values. This is addressed by modifying the formulas to include a negative skew factor that penalizes deficits (e.g., in concurrency and reliability) more than surpluses.

### 7.2 Rule-Based Planner

Once the CSP engine produces an optimal state, a sequence of steps is needed to transform the deployment environment. This is especially critical for in-mission planning since the CSP engine may request that a policy migrate from an over-provisioned device to a less utilized device. Policies with data in their queues or live sessions must be quiesced in an orderly fashion to prevent data loss or disconnected clients.

Hammerhead approaches the planning phase through the use of a rule-based implementation of a goal tree. The Drools working memory is prepared with an initial set of goals; e.g., *Deploy Policy A to Device 1*. Additional declarative rules decompose this goal into sub-goals, which are inserted into working memory recursively until all goal paths resolve to actions.

## 8 Implementation

The Hammerhead prototype is built using Java as a collection of web services and databases. Core components are implemented within a single Tomcat web service with dedicated endpoints for the ticketing system (to open and update tickets), bundle registry system (to load adaptation bundles from the database), SNMP management service (to register for notifications sent by BPD SNMP agents), and an Elasticsearch service (for retrieving traffic packets and their sizes). Each Hammerhead instance also contains a Node Agent, which is an embedded web service for communicating with other Hammerhead instances. A PostgreSQL database is used for storing adaptation bundles and tickets.

The constraint solver leverages the open-source Optaplanner project, which provides a clean separation between search heuristics, planning model, and scoring functions. Clients implement the latter two requirements as well as the detailed configuration and preferences for the heuristics. The functionality is encapsulated in a Planner module in Hammerhead. This includes a planner-specific model of the problem space, which includes 11 discrete and continuous attributes over hard and soft constraints. A future task, however, is to integrate the constraint solver into the adaptation bundle.

## 9 Evaluation

To verify system operation and response times for Hammerhead's core components, we created a continuous simulator in the context of a Blue Force Tracker system that processes location reports from two aircraft. As each BPD receives an aircraft location report, it forwards the report to a receiver that shows the aircraft's current location on a map. At each 1 second timeslice, the system triggers Hammerhead's core reasoning engine, which queries the Elasticsearch database to retrieve traffic information as low-level events and inserts the events into the working memory of the rule engine, which performs event aggregation and potentially triggers higher-level rules to adapt the system.

Traffic sent to the BPD is generated using simulated clients on separate threads that each submit an XML-based location report every 100 ms to a simulated BPD. The simulated BPD Java class models the domain, including security

policies, flows (with separate inbound and outbound queues) and filters. A notional filter is used that drops messages at regular configured intervals, accepting the remaining messages. A DNS daemon is used as a load-balancer to resolve BPD host names to available endpoints.

We can perturb the system by forcing a BPD device to drop all or a significant number of incoming location reports, starving the receiver of data. This effectively simulates a bad filter configuration. We can also introduce firewall rules that prevent the BPD from reaching its receiving network interface. This results in a growing outbound message queue on the flow, simulating a blocked flow.

Though the sending clients and the Hammerhead instances run in a single virtual machine, each communication endpoint is allocated its own network interface and JVM process inside separate Docker containers. This logical topology improves the fidelity of the simulation at the configuration level (i.e., we can configure TShark and other services as if the endpoints were physically separated).

The simulation shows the correct operation of the system under both conditions above, redirecting inbound flows to healthy BPDs in the pool. We use the simulator to gauge the feasibility of the core components' ability to monitor streaming network events, perform aggregation and reasoning, and trigger actuation, all within acceptable thresholds. The simulations show the feasibility of these components to respond within acceptable ($< 15$ seconds) response times for stream-oriented location data.

We evaluated the constraint solver using two techniques: Test based and simulation based evaluation. Test based evaluation focused on the initial assignment of policies to BPDs to ensure that all constraints were satisfied. The tests were limited to three BPDs and three policies. They produced expected results within 5 seconds on average.

The constraint solver was also verified using a discrete event simulator and workload trace from parts of the Google cluster management system (github.com/google/cluster-data), which provides data from a 12.5K-machine cell over a month long period in May 2011. The trace files provide several metrics that are also supported by the SNMP MIB (for which our BPD simulators include a compliant SNMP V3 agent). These metrics include sampled CPU usage across 5 minute averages, which was the focus of our experiments. Future experiments will investigate correlated free memory and other metrics. Each 1-second tick of the simulator provided a new CPU average to the constraint solver. We perturbed the baseline data using linear transformations to investigate the system behavior under different load averages and standard deviations. When the CPU load factor was not evenly balanced, the planner migrated a policy off the overloaded BPD onto a less busy device. One metric that will be explored in future work is a measure (penalty) for excessive policy migration (volatility).

## 10 Related Work

Vaquero, et al. survey several scalability solutions that feed data from sensors to a rule-based decision making module, which in turn evaluates network and server conditions, potentially scaling the environment through actuators (Vaquero, Rodero-Merino, and Buyya 2011). However, many existing systems limit scaling rules to individual VMs, versus supporting coordinated changes across multiple VMs and LBs. Another limitation is the solution insufficiently abstracts underlying infrastructure details, requiring service providers to maintain detailed VM configuration details. The most comprehensive approaches, however, like Claudia, provide a more flexible, abstracted solution that focuses on the application (Rodero-Merino et al. 2010). Hammerhead fits in this space, abstracting scalability to the data flow level and presenting a comprehensive sensor-control framework that coordinates changes across multiple servers and LBs. However, although Claudia includes some "smart scaling" rules, it lacks a general constraint satisfaction engine. Constraint-based solutions do exist (Jayasinghe et al. 2011), however their approach is specific to fixed types of structural constraints applicable to Infrastructure as a Service (IaaS) data centers.

Our contribution lies in the combination of techniques to bring automation to and increase the response speed of BPDs where information exchanges are restricted and administrators must understand and approve the automation and have run time visibility into the system's behavior.

## 11 Conclusion

The Hammerhead prototype leverages AI techniques – knowledge engineering, a rule-based system, and a constraint solver – to provide a predictable, understandable, and flexible automation solution to greatly improve the reliability and adaptivity of BPDs. As the commercial vendors move toward adopting the technologies in the Hammerhead prototype, the security and reliability of the BPDs will increase thus increasing the security and reliability of government networks allowing authorized sharing of information while preventing unauthorized sharing and attacks.

## References


Jayasinghe, D.; Pu, C.; Eilam, T.; Steinder, M.; Whalley, I.; and Snible, E. C. 2011. Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement. In Jacobsen, H.-A.; Wang, Y.; and Hung, P., eds., *IEEE SCC*, 72–79. IEEE Computer Society.

Rodero-Merino, L.; Vaquero, L. M.; Gil, V.; Galán, F.; Fontán, J.; Montero, R. S.; and Llorente, I. M. 2010. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems* 26(8):1226–1240.

Vaquero, L. M.; Rodero-Merino, L.; and Buyya, R. 2011. Dynamically scaling applications in the cloud. *SIGCOMM Comput. Commun. Rev.* 41(1):45–52.