Coordinated Management of Large-Scale Networks Using Constraint Satisfaction

Martin Michalowski and **Mark Boddy** and **Todd Carpenter**
Adventium Enterprises, LLC
111 Third Avenue South, Suite 100
Minneapolis, MN 55401 USA
{first.lastname}@adventiumenterprises.com

Abstract

In this paper, we describe a toolset for managing the configuration and management of large-scale networks. In particular, we focus on managing limited processing and communication resources for coordinated network cyber-defense applications. Our implementation encompasses the complete cycle, from initial network modeling and extraction of the relevant constraints, through translation into a formal constraint model, and finally the application of a Linear Programming solver to determine feasibility. This system has been demonstrated on realistic cyber-defense network models provided by domain experts, as well as on automatically-generated models, used to explore the scaling behavior of the system.

Introduction

Due to the scale and diversity of modern network architectures and the increasing range of missions being supported by those networks, current means for designing, fielding, controlling, and maintaining network-wide cyber-defense applications do not scale to real world applications. For example, the United States Air Force (USAF) must protect a large and diverse set of interconnected networks spanning from unstable, low bandwidth, weakly connected ad hoc tactical components (e.g., ground sensors) to more stable operational components (e.g., aircraft, ground control, ISR platforms) through to very large, stable strategic components (e.g., SATCOM). These networks support a broad range of missions with real-time, mission-critical, and life-critical requirements. Misconfigured defensive deployments that accidentally consume more resources than expected, make unplanned system modifications, or otherwise unfavorably affect mission performance can have severe consequences.

On a small scale, coordinated cyber-defense operations can be managed using carefully constructed deployment rules and controls. For networks consisting of hundreds of thousands of nodes, manual oversight and configuration is infeasible. In this paper, we describe a toolset for managing the configuration and management of large-scale networks. In particular, we focus on managing limited processing and communication resources for coordinated network cyber-defense applications. Our implementation encompasses the complete cycle, from initial network modeling and extraction of the relevant constraints, through translation into a formal constraint model, and finally the application of a Linear Programming solver to determine feasibility. This system has been demonstrated on realistic cyber-defense network models provided by domain experts, as well as on automatically-generated models, used to explore the scaling behavior of the system.

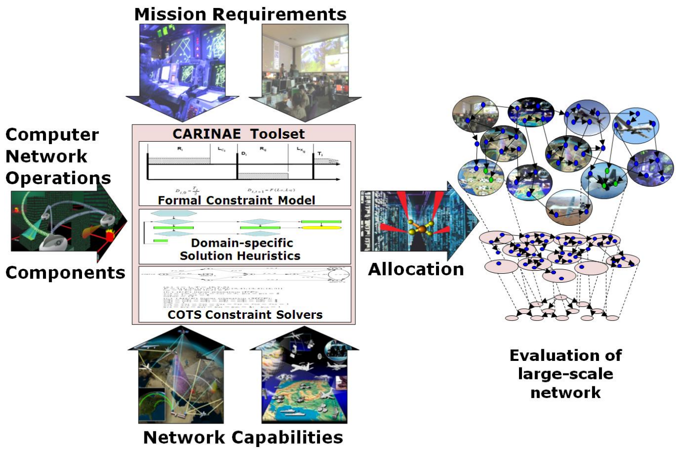


Figure 1: Cyber Architecture Reasoner Inferring Network and Application Environments (CARINAE)

Given a network model and information regarding current and planned network operations in support of both missions and network defense, the Cyber Architecture Reasoner Inferring Network and Application Environments (CARINAE) system shown in Figure 1 provides Cyber-Defense System developers and operators with the means to detect and resolve resource conflicts in network cyber-defense operations. Focused on large, service-oriented net-centric enterprise systems, CARINAE leverages constraint-based reasoning and open source, industry standard tools to create a robust analytical architecture that can analyze the interactions between network configurations and mission requirements for large-scale defensive cyber applications. CARINAE provides bandwidth, memory, and computational performance guarantees for large networks supporting diverse operational missions and defensive applications. CARINAE has been employed to analyze networks consisting of up to 1,000,000 nodes.

Motivating Example

Figure 2 shows a simple network model supporting a four-way video conference, where users on four different networks are using a video conference server a2. This video conferencing task places access, bandwidth, and quality of service (QoS) obligations on the satellite link S1, the firewalls protecting the individual networks, and the routers on the WAN. It also levies memory and processing obligations on the endpoint hosts, most notably a2.

Whether this task is feasible depends on the current state of the network. If S1 or any one of the routers becomes inoperable, or if one of the firewalls or routers is configured to deny access on the ports used for the video conference traffic, one or more of the participants will be unavailable. Similarly, it is possible that the satellite link, or some combination of links on the WAN will be bandwidth limited to the point where they cannot support the throughput required. The issue of bandwidth limitations becomes more of a concern when we introduce a second task. In this task, users on network B, C and D collaborate in performing a data fusion task, which levies its own connectivity and bandwidth obligations on the network infrastructure. If both this task and the previous video conference task are performed simultaneously, there may be infeasibility resulting from bandwidth constraints on one or more of the network links.

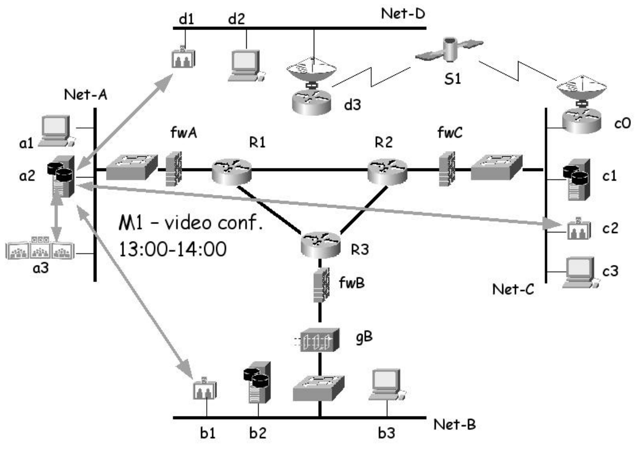


Figure 2: Video Conference Task

Constructing the Feasibility Model

Determining mission feasibility starts with the extraction of a mathematical statement of the feasibility problem from the mission and network models. More specifically, this process starts with a set of tasks to be supported and the network configuration that will be in place at the time those tasks are active. Each one of these tasks has an associated set of *resource requirements*, specifying the need for system resources for the duration of that task. In the video conferencing example shown in Figure 2, these resources might consist of bandwidth requirements from each of the remote locations to the video server, as well as processing time and memory requirements on both the server and the remote nodes. These bandwidth requirements may specify a particular route through the network (for example, an encrypted channel), a set of routes, or make no specification at all beyond the need for a certain level of throughput, however it is to be satisfied, in which case CARINAE may allocate that bandwidth across multiple paths through the network. For example, the bandwidth requirement associated with b2 can be realized by distributing the bandwidth across the direct route R3 to R1, and the less direct route through R2.

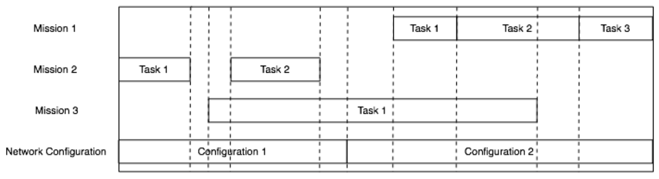


Figure 3: Example Mission Tasks and Network Configuration Timeline

As a result of this extraction process, we have a set of resource requirements, or *demands*, derived from the tasks associated with one or more planned missions, and a set of resource availabilities, or *capacities*, derived from the current network configuration. The tasks associated with each mission must be scheduled: each will have a specified starting and ending time. There may also be planned network configuration changes, which also take place at specified times. As shown in Figure 3, these times partition the timeline into periods over which both the set of tasks and current configuration are unchanging. We refer to this as a *multi-period* model. Multi-period models involving continuous allocations of bandwidth, CPU, and memory resources can be represented by sets of linear inequalities, and solved using *linear programming* (LP) solvers, which are capable of handling problem sizes involving hundreds of thousands of variables, and millions of constraints.

Implementation

The CARINAE implementation has several components, as well as a set of well-defined interfaces between them. The task and network models are maintained in the Architecture Analysis and Design Language, using the Eclipse-based Open-Source AADL Tool Environment (OSATE). From these models, an Eclipse plug-in is used to extract resource demands associated with tasks, and resource capacities as determined by the projected network configuration. The output of this plugin is formatted in the MINIZINC (Nethercote et al. 2007) constraint modeling language. The resulting constraint problem can then be submitted to the accompanying MINIZINC solver or other CSP solvers that accept the MINIZINC language.

However, both the MINIZINC solver and other solvers capable of accepting or translating MINIZINC input, such as

GeCode[1] are problematic for this application, based on their expressive limitations (for example, the use of floating-point numbers), or their failure to scale to very large problem instances. Consequently, our use of CSP solvers accepting MINIZINC directly was limited to small test cases, used to debug the constraint extraction and modeling process.

The scalability desired for CARINAE was a primary motivation for starting with a multi-period model, simple forms of which can be represented as linear programs, and more complex versions of which can be addressed either through repeated solutions of an LP, or using Mixed-Integer Linear Programming. Therefore we translate the MINIZINC representation of the problem into MPS, an input language accepted by a broad array of linear programming systems.[2] This model is then solved using CLP, an open-source C++-based linear programming solver.[3] CLP has proved remarkably efficient and robust, scaling effectively to network models involving over one million nodes.

This architecture has several advantages. First, it isolates the extraction of the feasibility model from the network and mission models, separating that process completely from the choice of solver and solver input format. Second, using MINIZINC as an intermediate representation provides considerable expressive flexibility. In addition to the LP models currently being employed, MINIZINC can represent finite-domain constraints, with built-in constructs supporting high-level specification of constraints. Thus, we can use the same intermediate representation while varying either or both of the network and mission models, or the solver and solver input language. This provides an ideal basis for exploring alternative modeling formalisms, languages, or techniques.

Constraint Model

Once extracted from the network and mission models, the feasibility model is represented as a set of *constraints* among a set of *variables*, corresponding to decisions about network configuration, mission schedules, and possibly choices among different means of achieving a particular mission. Additional variables represent the network's *state*, comprising its current (fixed) physical and logical structure, the set of active or scheduled missions, and the resulting network demands. For example, modeling the video conference task requires representing all of the hardware and bandwidth requirements specified, then solving to find an *assignment* of those demands to nodes or links that have available the required capacity. In the CARINAE model, we represent computing nodes as *processing elements*, which may be organized hierarchically. Communication between nodes is via a set of *links* among *ports*. Task resource requirements are represented as a set of *demands* on the available resources.

Definition of a CSP Model

We start by defining an instance of a CSP problem $\mathcal{C}$ as a tuple

$$\mathcal{C} = \langle \mathbf{E}, \mathbf{P}, \mathbf{L}, \mathbf{D} \rangle$$

comprising

- a set $\mathbf{E}$ of *processing elements*,
- a set $\mathbf{P}$ of *ports*,
- a set $\mathbf{L}$ of *links*, and
- a set $\mathbf{D}$ of *demands*.

For the work reported in this paper, we employ a *static* CSP model, which describes the state of the system at a single instant (or over a single period) of time. Consequently all resource limits and demands are expressed in time-free terms. CPU demand is expressed as a MIPS requirement. Communications demand is expressed as a requirement for a specified data-rate. Memory demand is expressed as an amount of memory that must be allocated, out of a finite store.

Processing Elements A processing element $e \in \mathbf{E}$ has the following attributes [4]

- CPU capacity (in MIPS): $\mathsf{mips}(e) \mapsto \mathbb{R}^+$
- memory capacity (in MB): $\mathsf{mem}(e) \mapsto \mathbb{R}^+$
- A set of sub-elements: $\mathsf{sub}(e) \mapsto \mathbf{E}$
- CPU-aggregate-utilization: $\mathsf{aggmips}(e) \mapsto \mathbb{R}^+$
- memory-aggregate-utilization: $\mathsf{aggmem}(e) \mapsto \mathbb{R}^+$

Ports Ports collect communication flows into and out of a particular processing element. A port $p \in \mathbf{P}$ has these attributes:

- an associated processing element: $\mathsf{pe}(p) \mapsto \mathbf{E}$
- throughput capacity, defined in Mbps: $\mathsf{Mbps}(p) \mapsto \mathbb{R}^+$

Links Communication connectivity is provided by *links*. Links are directional, with the following attributes for a link $l \in \mathbf{L}$:

- a source port: $\mathsf{src}(l) \mapsto \mathbf{P}$
- a destination port: $\mathsf{dest}(l) \mapsto \mathbf{P}$
- throughput capacity, defined in Mbps: $\mathsf{Mbps}(l) \mapsto \mathbb{R}^+$

We've chosen to use directed rather than undirected edges, and do not explicitly model hyper-edges (connections among larger sets of nodes than pairs).

Demands We have three resources, and so three kinds of demand. CPU and memory demands $d \in \mathbf{D}$ have the following attributes:

- a processing element with which the demand is associated: $\mathsf{orig}(d) \mapsto \mathbf{E}$
- a demand level: $\mathsf{demand}(d) \mapsto \mathbb{R}^+$

A communication demand $d \in \mathbf{D}$ has

- a *source* port: $\mathsf{src}(d) \mapsto \mathbf{P}$
- a *destination* port: $\mathsf{dest}(d) \mapsto \mathbf{P}$
- a demand level: $\mathsf{demand}(d) \mapsto \mathbb{R}^+$
- a set of *allowed links*: $\mathsf{allowed}(d) \mapsto 2^{\mathbf{L}}$

[1]www.gecode.org

[2]lpsolve.sourceforge.net/5.5/mps-format.htm

[3]projects.coin-or.org/Clp

[4]$\mathbb{R}^+$ denotes the set of non-negative real numbers.

The symbol $2^{\mathbf{L}}$ denotes the *power set* of $\mathbf{L}$. To differentiate among the different types of demands, we define subsets of $\mathbf{D}$: $D_{cpu},\ D_{mem},\ D_{comm}$, each comprising all of the CPU, memory, and communication demands, respectively.

Processing Element Constraints

Processing elements are defined in a part/whole hierarchy of elements and sub-elements. For processing elements e_i, e_j, where $i \neq j$:

- $e_i \in \mathsf{sub}(e_j) \Rightarrow e_j \notin \mathsf{sub}(e_i)$
- $e_i \in \mathsf{sub}(e_j) \Rightarrow e_i \notin \mathsf{sub}(e_k), \forall k \neq j$

In AADL terminology, we may have a node, which has multiple CPUs, which support multiple virtual machines (VMs), which support multiple platforms. The term processing element can be applied to either hardware or software. There are two possible views of the processing element hierarchy. In the *aggregate* model, processing elements at any level in the hierarchy impose constraints corresponding to usage attributes, interpreted as resource limits to be compared to their aggregate-utilization attributes. For processing elements having no sub-elements, the aggregate-utilization attributes are set directly (see the *Demand Constraints* section). For processing elements with sub-elements the aggregate-utilization attributes are computed from the aggregate-utilization attributes of their sub-elements.

$$\forall e \in \mathbf{E}, \mathsf{aggmips}(e) \leq \mathsf{mips}(e) \tag{1}$$

$$\forall e \in \mathbf{E} : \mathsf{sub}(e) \neq \emptyset, \quad \mathsf{aggmips}(e) = \sum_{e' \in \mathsf{sub}(e)} \mathsf{aggmips}(e') \tag{2}$$

Similarly for memory:

$$\forall e \in E, \mathsf{aggmem}(e) \leq \mathsf{mem}(e) \tag{3}$$

$$\forall e \in \mathbf{E} : \mathsf{sub}(e) \neq \emptyset, \quad \mathsf{aggmem}(e) = \sum_{e' \in \mathsf{sub}(e)} \mathsf{aggmem}(e') \tag{4}$$

This is a good model for aggregated global resources such as power or communication bandwidth, where at any level of the hierarchy the sum of the budgets for the next level down may be more than the capacity limit imposed (we assume not everyone will draw their maximum budget concurrently).

In the *budget* model, processing element capacities impose constraints both down the hierarchy (resource limits) and up the hierarchy (resource demands). In this case, we replace the constraints 2 and 4 above with

$$\forall e \in \mathbf{E} : \mathsf{sub}(e) \neq \emptyset, \quad \mathsf{aggmips}(e) = \sum_{e' \in \mathsf{sub}(e)} \mathsf{mips}(e') \tag{5}$$

$$\forall e \in \mathbf{E} : \mathsf{sub}(e) \neq \emptyset, \quad \mathsf{aggmem}(e) = \sum_{e' \in \mathsf{sub}(e)} \mathsf{mem}(e') \tag{6}$$

This is more appropriate for something like weight, or power budgets for sub-assemblies that don't get switched on and off. There is no requirement that either the aggregate or budget models be uniformly applied in a given processing element hierarchy; both may be needed, in different places.

Demand Constraints

We model two different types of demand constraints.

Memory and CPU Demands Memory and CPU demands are imposed by constraining the corresponding aggregate utilization for $\mathsf{orig}(d)$. For memory:

$$\forall d \in D_{mem}, \quad \mathsf{aggmem}(\mathsf{orig}(d)) = \mathsf{demand}(d) \tag{7}$$

and for CPU:

$$\forall d \in D_{cpu}, \quad \mathsf{aggmips}(\mathsf{orig}(d)) = \mathsf{demand}(d) \tag{8}$$

Additionally, we constrain $\mathsf{sub}(\mathsf{orig}(d))$ to equal $\emptyset$, restricting the imposition of memory and CPU demands to leaf elements in the processing element hierarchy. This does not restrict our ability to model demands at non-leaf nodes, because those demands can be assigned to a dummy leaf node which is then added as a sub-element of the appropriate processing element.

Communication Demands Communication demands are imposed by adding the required throughput to the specified ports. See Constraints 12 and 13, below.

Link Constraints

We can view the set of ports $\mathbf{P}$ and any set of links $L \subseteq \mathbf{L}$ in a given CSP model as a directed graph $G = \langle \mathbf{P}, L \rangle$, with vertices $\mathbf{P}$ and edges L, where each edge $l \in L$ is labeled with $\mathsf{Mbps}(l)$. Then G fits the definition of a *flow network*.[5] Because different communication demands are represented as different flows, with distinct sources and sinks, we need to represent this as a *multi-commodity* flow problem, with each demand d corresponding to a different commodity. Consequently, we define network *flow* on a specific link with respect to a given set of links L and demand d:

$$\mathsf{flow}_L(l, d) \mapsto \mathbb{R}^+$$

Then we add the following constraint:

$$\forall l \in L, \quad \mathsf{flow}_L(l) = \sum_{d \in D_{comm}} \mathsf{flow}_L(l, d) \tag{9}$$

The total flow on a given edge (link) must be less than the capacity:

$$\forall l \in L, \quad \mathsf{flow}_L(l) \leq \mathsf{Mbps}(l) \tag{10}$$

Demands can only flow on allowed links:

$$\forall l \in L, \forall d \in D_{comm} : l \notin \mathsf{allowed}(d), \quad \mathsf{flow}_L(l, d) = 0 \tag{11}$$

We define flow at a vertex (i.e., port) p, which also enforces conservation of flow in the network:

$$\forall d \in D_{comm}, \forall p \in \mathbf{P},\ \mathsf{flow}_L(p, d) = \sum_{\{l \in L \mid \mathsf{src}(l) = p\}} \mathsf{flow}_L(l, d) + \sum_{\{d \in D_{comm} \mid \mathsf{dest}(d) = p\}} \mathsf{demand}(d) \tag{12}$$

$$\forall d \in D_{comm}, \forall p \in \mathbf{P},\ \mathsf{flow}_L(p, d) = \sum_{\{l \in L \mid \mathsf{dest}(l) = p\}} \mathsf{flow}_L(l, d) + \sum_{\{d \in D_{comm} \mid \mathsf{src}(d) = p\}} \mathsf{demand}(d) \tag{13}$$

[5] `en.wikipedia.org/wiki/Flow_network`

These constraints add the communication demand, as well.

Note that according to this definition, there is no requirement that a given communication flow use a single path from one port to another. The throughput required may be spread over any or all of the possible paths between the two points. Finally, there are constraints on flows through ports, due to the capacities we allow to be specified on ports:

$$\forall p \in \mathbf{P}, \quad \mathsf{flow}_L(p) = \sum_{d \in D_{comm}} \mathsf{flow}_L(p, d) \tag{14}$$

$$\mathsf{flow}_L(p) \leq \mathsf{Mbps}(p) \tag{15}$$

Experimental Evaluation

For CARINAE, we demonstrated two somewhat-separate properties. First, we demonstrated the end-to-end *process*, starting with AADL representations of an assortment of network management problems related to cyber-defense. These problems were designed to reflect properties of real-world networks, and either generated by or validated by domain experts. The largest of these networks consisted of up to a million nodes, though most of our testing focussed on considerably smaller networks.

For the purposes of testing for scalability, we used an automated generator, so as to be able to more systematically control different properties of the networks being evaluated. The instances produced by our instance generator take the shape of a balanced tree where the user can control the depth and width of the tree. Additionally, the user can adjust the capacities of ports and the procedure used to generate the communication demands. Node connections in the tree are modeled using two communication links in opposite directions. Consequently, some path exists from any network node to any other node.

Linear Programming Solver Performance

In the graphs in this section, the x-axis shows the number of nodes in the problem instance, and the y-axis is the time required to generate a feasible solution.

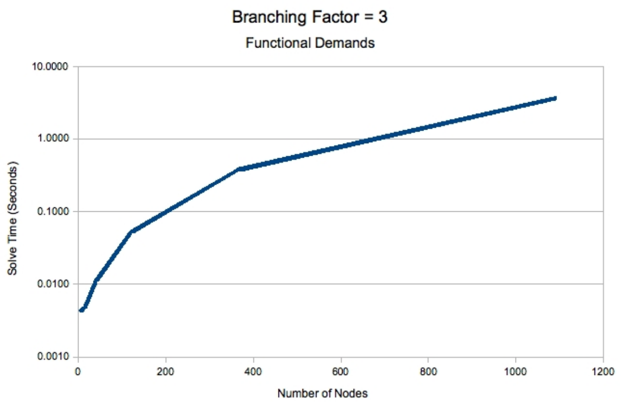


Figure 4: Branching Factor = 3, functional demands

Figure 4 shows the results obtained when using a constant branching factor of three for each problem instance and varying the depth of the tree from two through seven. In all cases, demands were automatically generated, so that each internal node in the network has at least one demand passing through it from one child node to another child node, in addition to any demands that require communication up or down the tree *through* that node. The number of demands in the network thus grows slightly faster than linearly.

These results demonstrate how the solving time increases as the size of the network grows by increasing the depth of the tree. In this configuration, the system is permitted to look for communication paths along any link in the subtree containing both the source and destination node. For nodes whose only common ancestor is the root, the demands can potentially use any link in the entire network.

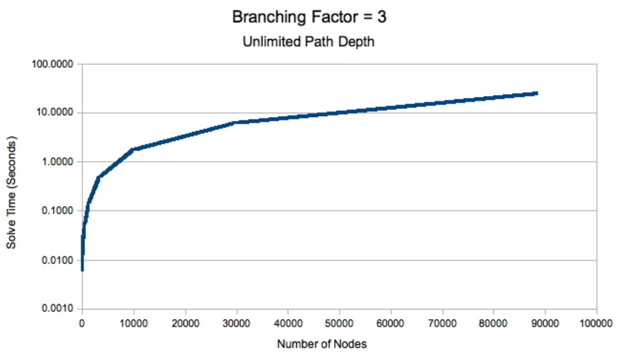


Figure 5: Branching Factor = 3, 20 demands

Figure 5 shows the growth in solution time with increasing network size for a *fixed* number of demands. Again, the branching factor is constant at three and the depth of the tree varies. A comparison to Figure 4 demonstrates that increasing demands as a function of network time makes the problem significantly larger and more difficult.

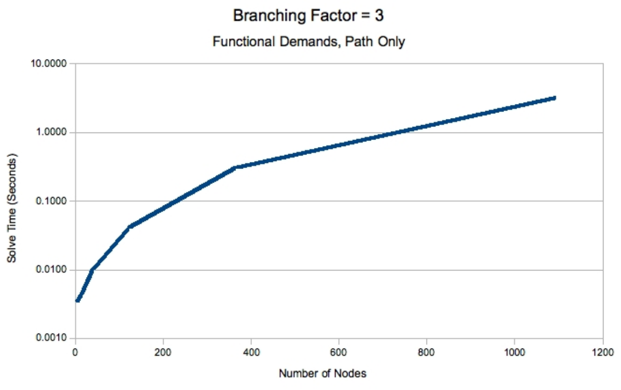


Figure 6: Branching Factor = 3, functional demands, path only nodes

In Figure 6, the instances are the same structure and size as those in Figures 4 and 5 but now the system will only consider paths using links either up or down the tree, imposing a unique path from one node to another. The performance improvement for this more restrictive model is minor, supporting the argument that the more flexible routing scheme does not lead to a significant extra cost for pairwise demands.

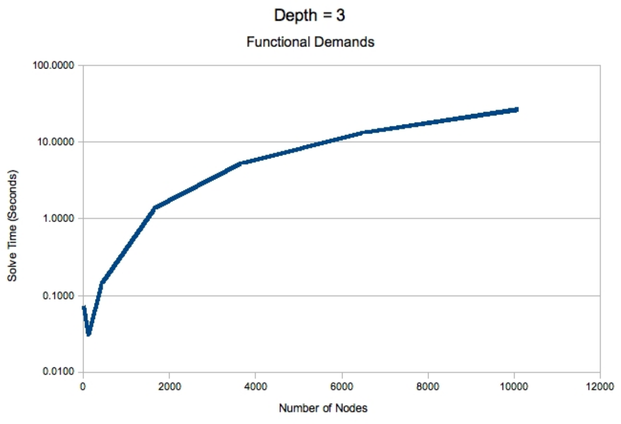
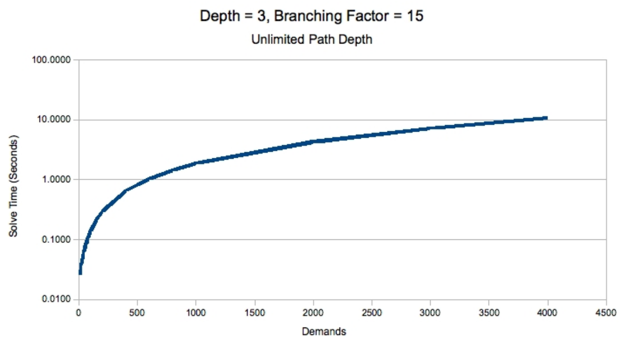


Figure 7: Depth = 3, functional demands

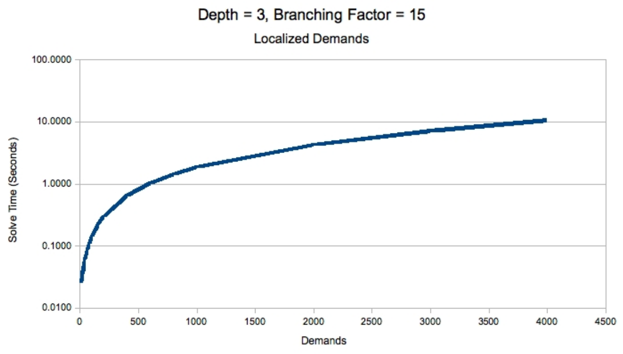
Figure 7 shows how solution time grows for a fixed-depth network with an increasing branching factor. In these sets of experiments we fixed the tree depth to three and varied the branching factor from three through 100. Again, the number of demands was functionally-based on the number of nodes, growing slightly faster than linearly, and we used the same characterization of allowed paths as the models in Figure 4. When compared to the results presented in Figure 4, we see how longer paths (deeper trees) versus more paths (wider trees) through a given router affects performance in this model. The solve time grows slightly faster in instances with more paths (wider trees).

To test the growth rate of the solution time as a function of the number of demands, we generated instances with a fixed depth of three and branching factor of 15, varying the number of demands from 10 through 4000. Figure 8(a) shows that very close to linear growth in solution time with respect to the number of communication demands (messages) for a fixed-size network. Comparing Figure 8(a) to Figure 8(b) shows that the solution time is not significantly affected by restricting demands to be between leaf nodes with a common immediate parent, despite the fact that this strongly restricts the possible paths between the two nodes.

Figure 9(a) and Figure 9(b) show performance information for a broadcast message. This is specifically NOT multicast: each network host is sent its own message. The performance in Figure 9(b) is significantly improved over Figure 9(a), by virtue of the routing guidance provided by restricting possible paths to network links up and down the tree. This is in dramatic contrast to the pairwise communication modeled in Figure 4 and Figure 6, where this restriction had only a minimal effect. Furthermore, this restriction puts the performance for a broadcast message in the same general area as for pairwise messages, with or without the path restriction.



(a) Depth = 3, Branching Factor = 15



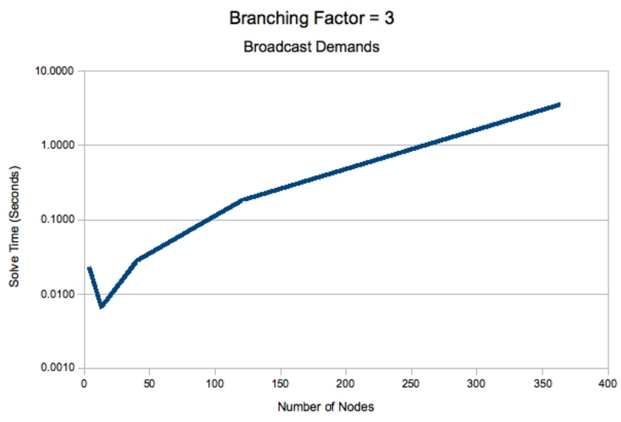
(b) Depth = 3, Branching Factor = 15, localized demands

Figure 8: Varying Demands in a Fixed Network

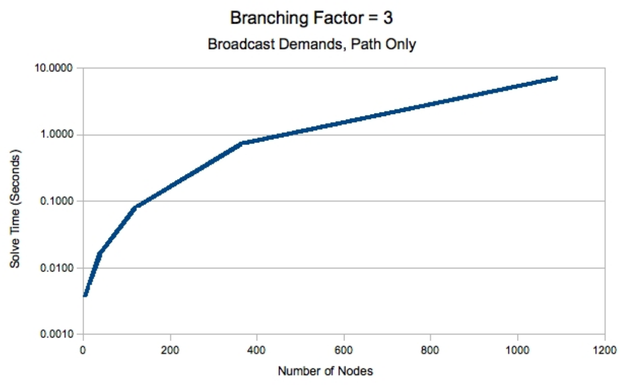
Related Work

The solution techniques we employ are drawn from the very large body of work on methods for Constraint Satisfaction and Constraint Optimization for combinatorial, continuous, and hybrid problems. See, for example (Dechter 1989; Nadel 1990; Boddy and Johnson 2002; Michalowski and Knoblock 2005; Hentenryck and Michel 2006). The work reported here uses purely continuous models, but in using MiniZinc, we have deliberately opted for a significantly more expressive constraint language. As discussed in the next section, we plan to extend CARINAE to handle more general problems, for which this expressive power may be needed. There is as well a considerable body of work on building network models, including the use of tools such as NMAP to construct these models largely or completely automatically. Deriving a CSP model of the form we employ here directly from a network model is not something we have previously seen discussed.

Tools for large-scale network configuration management are not widely available. The need for these tools shows clearly in the increase in propagation speed of network worms, such as Code Red and Sapphire/Slammer, early in this decade. Code Red I infected 359,000 Internet hosts on July 2001 in less than 14 hours (Moore, Shannon, and Brown 2002). Its infection rate peaked at 2,000 hosts per minute, and the number of infected hosts doubled approximately every 37 minutes. The following year, the Sap-



(a) Branching Factor = 3, broadcast demands



(b) Branching Factor = 3, broadcast demands, path only nodes

Figure 9: Broadcast Demands with a Constant Branching Factor of 3

phire/Slammer worm was two orders of magnitude faster, doubling every 8.5 seconds, and achieved its maximum infection rate of 55 million scans per second after only three minutes (Moore et al. 2003). Effective defense of large networks from attacks of this nature requires a coordinated response.

Discussion and Future Work

We have demonstrated that a multi-period feasibility model can be solved efficiently for very large instances. There are several directions in which we plan to extend this work. The first one is to support successively more flexible feasibility problems. For example, a simple scheduling problem can be supported by a minor extension to the multi-period model, in which missions are added one at a time, generating a small number of additional periods requiring solution. Scheduling problems that involve moving tasks around, or deciding whether or not to schedule missions, are potentially significantly more difficult to solve, because they contain a mix of both discrete and continuous variables.

In the work reported here, we have modeled mission requirements as independent demands on processing and communication resources. This puts the onus on the user to track two things. First, the user must coordinate demands (for example, the need to allocate both CPU and communication resources at the same time for the same task). Secondly, the user may need to specify sequential *phases* of the same mission. Business process modeling formalisms such as YAWL offer a way to capture a mission's tasks and the associated demands. The mission can then be analyzed, monitored or automated, with multiple demands thus being derived from a single representation (ter Hofstede et al. 2009).

Extending this model to consider latency as well as throughput is another direction for further work. For example, an important difference between Code Red and Sapphire/Slammer is that Code Red, which relied on a TCP connection to propagate, was limited by network *latency*, while Sapphire/Slammer, which consisted only of a single UDP packet, was limited by network *bandwidth* (Moore et al. 2003). In fact, Sapphire/Slammer's scanning quickly interfered with its own growth.

Finally, it is clear that defensive approaches will need to be more efficient than the malware that they combat. Theoretical analysis suggests that optimizations, such as deploying a list of target addresses and partitioning that list as deployment progresses, or implementing well-known servers to distribute scan lists upon request, can improve performance (Staniford, Paxson, and Weaver 2002), but the fastest possible deployments will rely on pre-determined "spread trees" to defend only known vulnerable hosts (Staniford et al. 2004). For example, the time to defensively inoculate N hosts with a K-way spread tree is projected as $O(\log_K N)$. Further scaling experiments modeling these more structured approaches to cyber-defense can provide useful information regarding the best areas for further work on either expressiveness or scalability for CARINAE.

Acknowledgments

This material is based upon work supported by the Air Force Research Laboratory under Contract Number FA8750-08-C.

References

Boddy, M., and Johnson, D. 2002. A new method for the solution of large systems of continuous constraints. In *Notes of the 1st International Workshop on Global Constrained Optimization and Constraint Satisfaction*.

Dechter, R. 1989. *Constraint Processing*. The MIT Press.

Hentenryck, P. V., and Michel, L. 2006. Nondeterministic control for hybrid search. *Constraints* 11(4):353–373.

Michalowski, M., and Knoblock, C. A. 2005. A Constraint Satisfaction Approach to Geospatial Reasoning. In *Proceedings of AAAI-05*, 423–429.

Moore, D.; Paxon, V.; Savage, S.; Shannon, C.; Staniford, S.; and Weaver, N. 2003. The spread of the sapphire/slammer worm.

Moore, D.; Shannon, C.; and Brown, J. 2002. Code-red: a case study on the spread and victims of an internet worm. In *Proc. of the 2nd ACM SIGCOMM Workshop on Internet measurement*, 273–284.

Nadel, B. A. 1990. Representation selection for constraint satisfaction: A case study using n-queens. *IEEE Expert: Intelligent Systems and Their Applications* 5(3):16–23.

Nethercote, N.; Stuckey, P. J.; Becket, R.; Brand, S.; Duck, G. J.; and Tack, G. 2007. Minizinc: Towards a standard cp modelling language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP2007)*, 529–543.

Staniford, S.; Moore, D.; Paxson, V.; and Weaver, N. 2004. The top speed of flash worms. In *Proc. of the 2004 ACM workshop on Rapid malcode*, 33–42.

Staniford, S.; Paxson, V.; and Weaver, N. 2002. How to own the internet in your spare time. In *Proc. of the 11th USENIX Security Symposium*, 149–167.

ter Hofstede, A. H. M.; van der Aalst, W. M. P.; Adams, M.; and Russell, N., eds. 2009. *Modern Business Process Automation: YAWL and its Support Environment*. Springer.