A Graph Based Model for Survivability Applications*

A. W. Krings
Computer Science Dept.
University of Idaho
Moscow, ID 83844–1010, USA
krings@cs.uidaho.edu

A. Azadmanesh
Computer Science Dept.
University of Nebraska at Omaha
Omaha, NE 68182–0500, USA
azad@unomaha.edu

Abstract

Many problems found in standard security and survivability applications can be transformed into graph and scheduling problems, thereby opening up the problems to a wealth of potential solutions or knowledge of limitations, infeasibility, scalability or intractability. This paper introduces a model to aid in the design, analysis, or operations of applications with security and survivability concerns. Specifically, a five step model is presented that transforms such applications into a parameterized graph model that, together with model abstraction and representations, can be the basis for solutions derived from graph and scheduling algorithms. A reverse transformation translates the solutions back to the application domain. The model is demonstrated using migratory agent security and fault-tolerant agreement and their transformation into chain constrained and group scheduling problems respectively.

Keywords: Scheduling, Security, Survivability, Transformation Model.

1 Introduction

Malicious attacks on computers and networks have reached epidemic proportions. Although much research has addressed the issue of increasing security of networked computer systems, problems and malicious acts are on the rise, rather than getting less [5]. Of special concern is the reliance of critical infrastructures on networked computer systems. The 1997 President's Commission on Critical Infrastructure Protection (PCCIP) "designated as critical

*Portions of this work were funded by grant #60NANB1D0116 from the National Institute of Standards and Technology, U.S. Dept. of Commerce.

certain infrastructures whose incapacity or destruction would have a debilitating impact on defense or economic security"[17]. Among the eight critical infrastructures identified were telecommunications, electrical power, gas and oil storage and transportation, transportation, and water supply. What these critical infrastructures have in common is that their underlying devices are controlled by communication networks and that their underlying physical infrastructure can be modeled by graphs. As such, it is reasonable to assume that many problems associated with these infrastructures may have solution spaces in areas that use graphs as common models, e.g. graph or scheduling theory.

In the area of cyber terrorism, computer and network security and survivability are the principal research areas addressing protection. Security is often viewed as addressing issues of confidentiality, integrity, availability, as well as accountability and correctness. Survivability, on the other hand, goes beyond security and has been formulated with respect to *Resistance* to, *Recognition* of, and *Recovery* from attacks, with a final iteration considering *Adaptation* [7]. Whereas resistance and recognition are typically associated with security, the main consideration of survivability is recovery. The recovery aspect can adopt many concepts from the area of fault-tolerance considering diverse fault models, which are directly affected by the topology and communication protocols of the systems involved.

The lack of success in securing networked computer systems may be attributable to the missing theoretical groundwork and mathematical models [8]. Most approaches to security and survivability are ad hoc. Thus, in the absence of standardized security test procedures claims, e.g. of intrusion detection systems, cannot be verified. Furthermore, it is not possible to compare relative results, as such comparisons would require a general common basis.

In an attempt to increase rigor in certain critical cyber problems, we are investigating transformation of security and survivability problems to other disciplines. Problem transformations in order to solve hard problems have been used extensively in mathematics and engineering. Well known examples include exponentiation or Laplace transformation. The general strategy is to transform the original problem into a different problem space in which known solutions exist, or solutions can be found at lesser cost. After a solution has been derived in the new problem space, a reverse transformation is used to translate the solution found back to the original problem space.

This research presents a transformation model to formalize certain survivability and security problems. The transformation model enables solutions to be based on graph and scheduling theoretical concepts. Section 2 gives a model overview. Section 3 explains the model using examples of graph and scheduling problems. Section 4 outlines two case studies in which the original problems are transformed to scheduling problems. Finally, Section 5 concludes the paper.

2 Model Overview

The basic philosophy of the transformation model is shown in Figure 1. We will first describe the model in general and will demonstrate it using a mobile agent and a distributed agreement application in Section 4. For the description of the model overview imagine that the application under consideration is associated with a general network of computers, or a critical infrastructure such as the electric power grid together with the data communication network controlling its devices.

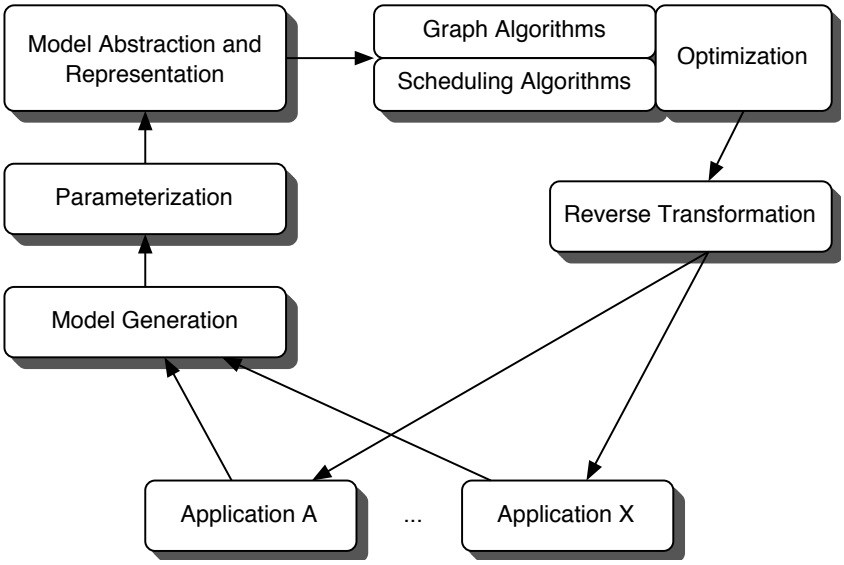

Figure 1: Model Overview

2.1 Model Generation

The application is transformed into a task graph together with the task model specification, if applicable. The general model is based on a directed graph $G = (V, E)$, where V is a finite set of vertices v_i and E is a set of edges e_{ij}, $i \neq j$, representing precedence relations between $v_i, v_j \in V$.

2.2 Parameterization

Now that the application is mapped to vertices and edges of G, a mapping of application specific parameters to generic parameters is needed. Examples of such parameters are power transmission, network throughput, communication cost, sensitivity or confidentiality, relative importance based on the cost of loss of services etc. The vertices and/or edges of

the graph generated need to be assigned weights representing their characteristics. The results can be generalized by integer or real valued weights. Thus, for each vertex in V and edge in E, vertex and edge weights are defined respectively. Let w_i^v denote the vertex weight of v_i. Furthermore, let w_{ij}^e denote the weight of edge e_{ij}, where v_i, $v_j \in V$ and $i \neq j$.

If multiple parameters need to be considered simultaneously, scalar weights are insufficient. Depending on the application that G represents, multiple weights may be defined for vertices and/or edges. In this case w_i^v and/or w_{ij}^e are weight vectors, where $w_i^v[k]$ and $w_{ij}^e[l]$ represent the k^{th} and l^{th} parameter respectively. The number of weights, representing the length of the vector, is denoted by $|w_i^v|$ and $|w_{ij}^e|$.

2.3 Model Abstraction and Presentation

Once a weighted graph G is defined, the graph can be considered in the context of standard graph or scheduling problems. A graph theoretical formulation can be represented by the graph itself, along with the manipulative objectives, such as max-flow or min-cut. On the other hand, a scheduling theoretical formulation requires the specification of the scheduling model, i.e. the processing environment, and the optimization criteria. In order to avoid lengthy descriptions of scheduling models S, a compact description of the form $S = (\alpha|\beta|\gamma)$ is commonly used [4]. The fields α, β, and γ indicate the processor environment, the task and resource characteristics, and the optimization criteria respectively. The most important feature of the model generation process is the matching of the survivability requirements and objectives with the graph and scheduling model and objectives.

2.4 Graph and Scheduling Algorithms

Graph G and schedule model S are now subjected to graph and scheduling theoretical algorithms respectively. The goal is to find optimal or suboptimal solutions for the sought after survivability criteria, applying the best suitable algorithm(s). A wealth of algorithms and heuristics of varying space and time complexity exist. Appropriate algorithms need to be identified that suit the optimization criteria, i.e. the survivability criteria, considering response time or computation requirements. One of the desired aspect of using graph or scheduling models is that the time or space complexity may be inherited from the algorithms, i.e. many problems have been shown to be intractable, e.g. NP-complete or NP-hard. This may provide valuable information about the solution space. However, it should be noted that intractability in the general case does not necessarily imply that the problem cannot be solved efficiently. In fact, for specific limited problem sizes solutions may be obtainable efficiently or at acceptable cost, despite of the problem being computationally hard. After the application of graph or scheduling algorithms or heuristics, optimal or sub-optimal solutions will be available.

2.5 Reverse Transformation

The solutions of the graph or scheduling algorithms must now be translated back to the application. This requires a reverse transformation analogous to the transformation used in the Model Generation. This step represents the transformation from the solution space back to the application space.

2.6 Application Adaptation

The model of Figure 1 can be used in an adaptive way by applying it in a circular fashion. Thus, after the reversed transformation the application could be refined, and the entire transformation process can be repeated utilizing the results of the previous iteration. Such iterative approach could benefit applications which are inherently dynamic and need the ability to adapt.

3 Applications

The process outlined above will now be explained using examples for a graph and scheduling problems.

Assume we need to analyze the vulnerability of a data communication infrastructure. The communication network may be represented by a digraph with weights equal to a specific Quality of Service (QoS) parameter, e.g. maximal data rate. The objective may be maintaining a desired data rate, even if some links fail. Such scenario may arise when a communication infrastructure is to be analyzed with respect to its resilience to malicious acts destined to disrupt communication links.

In this scenario the graph G is defined by the network graph. Vertices in V constitute communication hardware, e.g. gateways or routers, and edges in E are communication links. The QoS parameter, the maximal data rate w^e_{ij}, is defined for each link between devices v_i and v_j.

The scenario above is a graph problem which can be formulated as follows: Given a minimal required data rate between any two vertices v_i and v_j, find the minimal number of network links that must be destroyed in order to violate the data rate requirement. The answer should provide insight about vulnerabilities, and the minimal attack scenarios could be used to motivate enhancement to the robustness against attacks.

To demonstrate how security problems can benefit from solutions of their transformed scheduling problems, consider a system facilitating autonomous migratory agents to perform security related operations, e.g. patch management or diagnostics. Assume that, as a result

of an exposed vulnerability, patches need to be automatically installed in a large set of computers in such a way that patch management does not affect the overall mission.

The patch management agents will have a set of core tasks to be performed, diagnosing the presence of a certain vulnerability and installing the associated patch. As patch management often implies disruption of normal operations, including reboot of the computer, each computer needs to be queried, identifying the earliest time r_i at which the patch may be installed as well as the disruption weight w_i on computer v_i. If a patch management agent is viewed as a processor, and diagnosing and installing the patches on a specific computer v_i as a task T_i with a minimum and maximum processing time p_i^{min} and p_i^{max} equal to the diagnosis and patch installation duration respectively, then the agents paths may be determined by solving the scheduling problem that optimizes the makespan C_{max}, $\sum C_i$ or $\sum w_i C_i$.

3.1 Processors and Tasks

Whereas in scheduling resources are usually seen in the traditional sense, e.g. computers and machines, in the field of security and survivability resources may be interpreted more generally. There are many different attributes associated with different resources. For example, processors may be identical, uniform, unrelated or dedicated [4].

1. *Identical*: homogeneous environments, e.g. homogeneous computers and input/output devices, software licenses, personnel with same skill level.
2. *Uniform*: processors with different *speed* b_i, e.g. heterogeneous computers and input/output devices, network devices with different bandwidth, personnel at different skill levels within the same expertise domain, network sniffers with different sniffing capabilities, and vulnerability analyzers with different capabilities.
3. *Unrelated*: in classical scheduling theory, uniformity implies that processor speeds differ but individual processor speeds are considered constant. If the speed is dependent on the task performed, processors are called *unrelated* [4]. Typical examples are computers or networks subjected to Denial of Service (DoS) attack, which may be distributed (DDoS).
4. *Dedicated*: specialized for the execution of certain tasks. Examples include special purpose computers, or domain specialists with different areas of specialization, e.g. Unix and NT system administrators.

Tasks may be non-preemptive or preemptive.

1. *Non-preemptive*: once a task is started, its execution can *not* be interrupted. The classical example is printing. However, security policies often have strong require-

ments for patch management, redundancy management, and maintenance operations such as backups and logging that represent atomic, i.e. non-preemptable, operations.

2. *Preemptive*: task execution can be interrupted. This is standard for the majority of applications running on most commercial general purpose and real-time operating systems. Besides traditional interpretations, security policies specify the course of action in response to many kinds of benign and malicious activities, thereby preempting normal system operations. In response to system problems or attacks, technical response assistance centers require technicians to frequently process several incidences at a time, switching back and forth between them.

3.2 Model Mappings

There are countless security problems that can benefit from applying the model of Figure 1. Below we identify some problems, but it should be noted that this is by no means an attempt at generating a complete list of application domains.

Graph Model Many Critical Infrastructure Protection (CIP) problems have topology maps that can be represented by directed or undirected graphs. Typical examples are transportation networks, electrical power grids, pipelines, water lines, and the communication networks controlling these infrastructures. Below are some examples that would result in a graph model.

1. With respect to determining emergency procedures involving the transportation infrastructure, identification of the number of suitable disjoint paths between two locations, e.g. cities, and the associated traffic capacities are essential.

2. The identification of prime targets, whose destruction or compromise would render the application useless is a key task in CIP and communication networks. The 1997 report of the Presidential Commission on Critical Infrastructure Protection (PCCIP) identified infrastructure elements that are essential to the national defense and economic security of the United States.

3. Identification of the minimal infrastructure configuration needed to satisfy basic requirements. The definition of "basic" could differ depending on the application, e.g. military vs. civilian infrastructures.

4. After outages in the electric power infrastructure, it is very common for the maintenance personnel to physically reset *Supervisory Control and Data Acquisition* (SCADA) devices, rather than resetting or reactivating the devices remotely. Directing the maintenance crews to power substations is heavily influenced by the significant geographic distances between substations. Solutions may be derived from solving the associated traveling sales person problem.

Scheduling Model Many security problems can be mapped to scheduling problems. One key observation is that security personnel or mechanisms can be viewed as the processor environment. There are many ways to map the security problem examples described below to a scheduling problem $\alpha|\beta|\gamma$. The formulations shown are only examples. Especially in the task and resource characteristics, i.e. β, many interpretations and formulations are possible. This vagueness should be seen as a strength, not weakness, as it introduces flexibility in generating a solution space.

1. Response management: $P||C_{max}$.
   Each of the P identical processors represents a security specialist.

2. Response management: $P|r_i|C_{max}$ or $P|r_i, d_i|\sum w_i D_i$.
   These are variations of the previous case, where release times r_i, due dates d_i, and tardiness D_i are considered. Schedule the security tasks, as they arrive (perhaps with dynamic arrival times) under considerations of deadlines.

3. Response management: $Q|r_i|C_{max}$ or $Q|r_i, d_i|\sum w_i D_i$.
   In this variant, the realistic assumption is taken that the security personnel has different levels of expertise, e.g. junior versus senior security officer. Thus personnel is represented by *uniform* processors Q.

4. Response management: $J||C_{max}$.
   If the assumption is that different personnel has different qualifications, e.g. specialization with respect to specific operating systems, then processors might be considered to be *dedicated*. In this case the problem might be treated as a job shop.

5. Response management: $Pm|r_i|C_{max}$ or $Pm|r_i, d_i|\sum w_i D_i$.
   Given a specific task to be performed in a time critical application, determine the suitable number m of specialists needed for a given objective.

6. Response management: $P|pmtn|C_{max}$ or $P|pmtn, r_i, d_i|\sum w_i D_i$.
   Preemptive scheduling ($pmtn$) considers context switching time and the number of preemptions. This could include minimizing the number of preemptions. Technical support staff of most customer response centers subject their technicians to multiple cases at a time.

7. Physical security: $P|prec, group, r_i, d_i|\sum w_i D_i$ or $P|chain, group, r_i, d_i|\sum w_i D_i$.
   A facility is monitored by security cameras and observation personnel [11]. Different locations to be observed are represented by task groups G_i, where G_i consists of n_i tasks, $T_{i1}, T_{i2}, ..., T_{in_i}$. Each task T_{ij} represents an observation window of processing length p_{ij}. The processors are the observation monitor(s), or observation person(s). Groups are to be scheduled with deadlines less than the times required by a skilled adversary to infiltrate the facility. Solutions would be based on group scheduling or scheduling with *strong precedence* under consideration of precedence ($prec$).

8. Physical security: $F||C_{max}$ or $J|d_i|\sum w_i D_i$.
   The previous physical security application can also be formulated as a flow shop problem, capturing the requirement that $T_{j\ i-1}$ needs to be executed before T_{ji}, or as a job shop problem.

9. Agent security: $1|r_i|C_{max}$ or $Pm|d_i|\sum w_i D_i$.
   In systems facilitating autonomous agents to perform security related operations, e.g. version tracking, diagnostics, etc., the agent might have a set of critical tasks to be performed on specific systems [12]. In the case of distributed denial of service attacks (DDoS), timing might be crucial when implementing survivability measures using the agent, as the defensive measures race in time against the increasing affects of the DDoS attack. The agent is represented by a processor, the systems to be traversed are the tasks.

10. Agent security: $R|r_i|C_{max}$ or $Rm|d_i|\sum w_i D_i$.
    The previous case can be viewed with respect to the affects of the DDoS attack. In this case it might be valid to consider unrelated processors. Recall that the speed of an unrelated processor depends on the the particular task processed [4].

11. Agent security: $1|r_i, p_i|C_{max}$ or $Pm|d_i|\sum w_i D_i$.
    Similarly, if agents are used for patch management, installing patches has to be coordinated with the usage of the system in order to avoid conflicts with the tasks executing on the system. This is very obvious in operating systems that require applications to be terminated before installing the patches or rebooting after installation.

12. Intrusion detection systems: $P|r_i, p_i|C_{max}$ or $Pm|p_i, d_i|\sum w_i D_i$.
    Many intrusion detection systems (IDS) rely on centralized computers to collect data from log files and audit traces of different clients in order to check for coordinated attacks. The individual log files can be large in size and need to be downloaded in local area networks (LAN) or wide area networks (WAN).

13. Attack recognition systems: $1|p_i, \tau|C_{max}$.
    Certain low-level attack recognition systems [13] need to be scheduled with end-to-end and real-time requirements, guaranteeing that they are instantiated at regular intervals τ. The periodic tasks are 1) sensor data collection and 2) attack signature evaluation.

4 Case Studies

The previous section presented examples of applications and indicated possible transformation problems. However, it was not shown how the applications can be systematically transformed. In this section we outline transformations, by applying the model of Figure 1 to two different problems that are at the core of some survivability, security, and fault-tolerance concerns.

4.1 Autonomous Mobile Agents

Due to the very nature of the concept of software agents, mobile agents have been the source of many security concerns. Malicious agents could impact computer survivability and malicious computers could compromise agent security and survivability. The concept of resistance, recognition and recovery can therefore be seen from a computer system's or agent's point of view.

This case study utilizes the research presented in [15] and considers the first case, i.e. we consider survivability aspect due to potential malicious agents. In order to address security concerns and system survivability we assume that agent systems utilize redundancy. Redundancy in agent system has been the focus of recent research [9, 10, 18, 19, 20], and our specific focus is on secret sharing [21, 19] rather than pure spatial or information redundancy. In secret sharing the information is divided into k shares, and one has to have all shares to use the information. If each agent carries a single share, secret sharing using k shares can be easily extended to achieve survivability by extending it to an *m-of-k* scheme, in which the information of m out of k uncorrupted agents is needed to perform an action [19]. We now apply the process of Figure 1.

Application: Assume a multi-agent system is used to implement survivability functionalities on a set of networked computer systems. The relative importance of each computer to the system is prioritized, e.g. indicated by an integer numbered priority index scheme. Next, assume that a secret sharing scheme is implemented requiring k_i shares for a specific processing element, i.e. computer, PE_i. Thus, all k_i agents must be present at computer PE_i to perform their specific function. Given this secret sharing agent survivability scheme, it is important to find efficient agent traversal paths, e.g. paths that maximize the efficiency of an agent system charged with reactionary response to malicious act.

Model Generation: The application is translated into a task graph $G = (V, E)$ consisting of K job chains C_k, each representing a group of computers. Each computer to be traversed by the agents is shown as a vertex, i.e. job, $J_{k,i} \in C_k$, for $1 \leq k \leq K$. The position of $J_{k,i}$ in C_k, i.e. the chain position, indicates the relative computer priority. Thus, if $Pr(PE_i)$ and $Pr(PE_j)$ denote the priority of PE_i and PE_j respectively with priority values in the range $1 \leq i, j \leq m_k$, then edge $e_{i,j}$ is introduced in E if $Pr(PE_i) = Pr(PE_j) - 1$. Edge $e_{i,j}$ represents job precedence $J_{k,i} < J_{k,j}$.

Parameterization: The parameters reflecting the functionality executed by the agents during job $J_{k,i}$ need to be specified. The job priorities are implicitly defined by the position of $J_{k,i}$ in chain C_k, and the processing and release times of $J_{k,i}$ are determined by $p_{k,i}$ and $r_{k,i}$ respectively. Potential cost or liabilities due to malicious act can be modeled by weight $w_{k,i}$.

Model Abstraction: Suitable scheduling models can be found in [6], where 'Strong'-'Weak' chain constrained scheduling is discussed. The specific model considered is

$$Pm|fix_j, chain, p_j = 1|C_{max},$$

where m is the number of machines (represented by agents) in the system. The number of machines necessary to process a job is indicated by fix_j, and *chain* indicating chain precedence. It should be noted that fix_j indicates the degree of secret sharing.

To show how this model suits the application an example from [6] given. Let J denote a partition of all jobs $J_{k,j}$. Furthermore, let J^{expr} denote the set of jobs that require all machines with indices indicated in *expr* for execution. Now jobs can be partitioned by their resource requirements. If one assumes $m = 3$, then jobs can be partitioned as

$$J = \{J^{123}, J^{12}, J^{13}, J^{23}, J^{1}, J^{2}, J^{3}\}. \tag{1}$$

The superscript indicates the machines required, e.g. J^{123} comprises all jobs that need machines M_1, M_2 and M_3 to execute, whereas jobs in J^{12} only require M_1 and M_2. The number of indices listed in the superscripts indicate the degree of secret sharing required by each of the jobs of the associated set. For example, each job in J^{123} requires that three agents need to be present before their respective functionality can be executed. Similarly, each job in J^{12}, J^{13}, and J^{23} require two agents, however, each set has its specific set of associated agents. The problem

$$P3|fix_j, chain, p_j = 1|C_{max}$$

captures the notion of scheduling the tasks in J.

Scheduling Algorithm: It is shown in [6] that the problem $Pm|fix_j, weak\ chain, p_j = 1|C_{max}$ can be solved in $O(n)$ time if all jobs of chain k belong to the same partition in equation (1). The last constraint is called *agreeable machine configuration*. The weak chain constrained implies that scheduling two jobs of a chain, under consideration of the chain precedence constraint, may be interleaved with other jobs. This interleaving is not allowed in *strong chain* constrained scheduling.

Problem $Pm|fix_k, strong\ chain, p_j = 1|C_{max}$, for $m > 1$ has been shown to be NP-hard in the strong sense [6]. Similarly, problem $P3|fix_j, strong\ chain, p_j = 1|C_{max}$ with agreeable machine configuration is NP-hard in the strong sense [2, 3].

Reverse Transformation: A valid schedule produced by any scheduling algorithm directly reflects the agent traversal path.

4.2 Case Study: Distributed Agreement

The next case study outlines the transformation for the problem of reaching agreement in the presence of faults. The specific application, derived from [14], is defined as follows.

Application: Consider the formulation of the well known "Byzantine General Problem" together with recursive algorithm $OM(t)$ presented in [16]. $OM(t)$ guarantees to achieve agreement using $N \geq 3m + 1$ processors, where m is the maximum number of maliciously faulty processors. The algorithm is based on rounds of message exchange. In each round each processor forwards values received in the last round to all other processors that have not receive it before. Agreement is achieved using majority calculations at each processor after the algorithm executes for $m + 1$ rounds.

Model Generation: In the recursive $OM(t)$ algorithm each processing node keeps track of values received in a so-called EIG-Tree [1], where each level represents a round of message exchange. This EIG-Tree can be mapped into an *agreement task graph* [14] $G^A = (V^A, E^A)$. The tasks in set V^A represent computation and communication tasks associated with the generation of the EIG-Tree and the final determination of the agreement value, which is a majority voting process. Edge set E^A defines the precedence constraints.

Parameterization: In this example, parameterization first implies the trivial mapping from each vertex $v_i \in V^A$ to task T_i, i.e. each v_i is a task T_i. Next, the processing time p_i of each T_i needs to be specified. If T_i is a computational task, then p_i is the processing time required to receive the messages and manipulate the data structures representing the EIG-Tree. The leaf nodes of the graph also perform the final voting, with the voted-upon value constituting the *agreement value*. Exact values for p_i can be either computed from the program segment or based on measurement. If T_i is a communication task, then p_i represents the overhead of the protocol stack, network interface, and data link. In this case it is more practical to use measurement for the determination of p_i.

Model Abstraction and Presentation: In [14] it has been shown that the performance of agreement algorithms is linked to how efficient certain task groups can be scheduled. The groups were defined to contain all tasks of the agreement task graph $G^A = (V^A, E^A)$ that are associated with a specific level of the EIG-Tree. Thus if the tree has k levels, then k task groups can be identified in G^A. The task groups of G^A must be scheduled on N processors, together with the standard application tasks of each processor, represented by graph $G_i^P = (V_i^P, E_i^P)$. The final scheduling model is a hybrid model which requires the scheduling of G^A to coincide with the static scheduling of each G_i^P. The resulting scheduling formulation is thus a variation of $PN|prec, group, static|C_{max}$, assuming a homogeneous processing environment, the agreement task graph G^A with identification of the task groups,

as well as static task to processor assignments within a group, in addition to application task graph G_i^P.

Scheduling Algorithm: The problem $Pm|prec, group, static|C_{max}$ can be solved with any heuristic with special focus on group or strong precedence scheduling. An optimal or suboptimal solution could be acceptable.

Reverse Transformation: The scheduler and dispatcher satisfying the static allocation scheme need to be implemented. Thus, the reverse transformation in this application is simply the implementation of both.

5 Conclusion

This paper presented a model that can be used to derive solutions from other areas, e.g. graph or scheduling theory, to solve problems occurring in security and survivability applications via problem transformation. The specific targets were applications that can be reduced to graphs or task systems to be scheduled under specific scheduling models. The model was demonstrated using two case studies. The problem of migratory agent security was transformed to chain constrained scheduling based on [15], and distributed fault-tolerant agreement was expressed as a group scheduling problem based on [14].

We hope that through this contribution, researchers from the areas of scheduling theory and operations research will realize that many problems in security and survivability could have potential solutions in their research areas.

References

[1] P. Berman, J. A. Garay, and K. J. Perry, "Optimal Early Stopping in Distributed Consensus", *Proc. 6th International Workshop on Distributed Algorithms (WDAG '92)*, LNCS 647, Springer-Verlag, Nov. 1992, 221-237.

[2] J. Blazewicz, P.W. Dell'Olmo, M. Drozdowski, and M.G. Speranza, "Scheduling Multiprocessor Tasks on Three Dedicated Processors", *Information Processing Letters* 41, pp. 275-280, 1992.

[3] J. Blazewicz, J., P.W. Dell'Olmo, M. Drozdowski, and M.G. Speranza, "Scheduling Multiprocessor Tasks on Three Dedicated Processors: Corrigendum", *Information Processing Letters* 49, pp. 269-270, 1994.

[4] Blazewicz, J., et.al., "Scheduling Computer and Manufacturing Processes", Springer-Verlag, 1996.

[5] CERT/CC Statistics 1988-2002, CERT Coordination Center, http://www.cert.org/stats/cert_stats.html.

[6] M. Dror, Kubiak, W., and Dell'Olmo, P., " 'Strong' - 'Weak' chain constrained scheduling," *Ricerca Operativa*, Vol. 27, 1998, pp. 35-49.

[7] E. Ellison, L. Linger, and M. Longstaff, *Survivable Network Systems: An Emerging Discipline*, Carnegie Mellon, SEI, Technical Report CMU/SEI-97-TR-013, 1997.

[8] Keynote Speech of the Information Survivability Workshop, part of the International Conference on Dependable Systems and Networks, DSN-2001, by Roy Maxion, CMU, Goteborg, Sweden, 2001.

[9] D. Johansen, et al., Operating System Support for Mobile Agents, *Proc. 5th IEEE Workshop on Hot Topics in Operating Systems*, 1995.

[10] D. Johansen, et al., NAP: Practical Fault-Tolerance for Itinerant Computations, Technical Report TR98-1716, Department of Computer Science, Cornell University, USA, November, 1998.

[11] A.W. Krings, and M.A. McQueen, "Distributed Agreement in a Security Application," *28th International Symposium on Fault-Tolerant Computing, Digest of FastAbstracts: FTCS-28*, IEEE Computer Society Press, Munich, Germany, June 23 - 25, 1998, pp. 37-38.

[12] A.W. Krings, et al., "A Two-Layer Approach to Survivability of Networked Computing Systems", *Proc. International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, L'Aquila, Italy, Aug 06 - Aug 12, pp. 1-12, 2001.

[13] A. Krings, et al., *Attack Recognition Based on Kernel Attack Signatures*, Proc. International Symposium on Information Systems and Engineering, Las Vegas, pp. 413-419, 2001.

[14] A. Krings, et al., *Scheduling Issues in Survivability Applications using Hybrid Fault Models*, to appear in Parallel Processing Letters.

[15] A. W. Krings, "Agent Survivability: An Application for Strong and Weak Chain Constrained Scheduling", to appear as paper STSSM01, 37^{th} *Hawaii International Conference on System Sciences*, (HICSS-37), Minitrack on Security and Survivability in Mobile Agent Based Distributed Systems, January, 2004.

[16] L. Lamport, M. Pease, R. Shostak, The Byzantine Generals Problem, *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, July 1982, 382-401.

[17] *Critical Foundations, The President's Commission on Critical Infrastructure Protection*, Government Printing Office, Washington, DC, Oct. 1997.

[18] K. Rothermel, and M. Strasser, A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents, *Proc. IEEE Symposium on Reliable Distributed Systems* (SRDS'98), West Lafayette, USA, October, 1998, pp. 100-108.

[19] F.B. Schneider, Towards Fault-tolerant and Secure Agentry, *Proc. of the 11th International Workshop on Distributed Algorithms* Saarbrucken, Germany. September, 1997.

[20] F.M. Assis Silva, A Transaction Model based on Mobile Agents, PhD Thesis, Technical University Berlin, 1999.

[21] Jay J. Wylie, et al., Selecting the Right Data Distribution Scheme for a Survivable Storage System, Technical Report, CMU-CS-01-120, Carnegie Mellon University, May 2001.