UI-CS-TR-02-024

A Graph Based Model for Survivability Analysis

by

A.W. Krings and M.H. Azadmanesh

August 2002

COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF IDAHO

A Graph Based Model for Survivability Analysis[*]

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

A. Azadmanesh
Computer Science Dept.
University of Nebraska at Omaha
Omaha, NE 68182–0500
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 problem to a wealth of potential solutions or knowledge of limitations, infeasibility, scalability or intractability. This report 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 a distributed agreement and its transformation into a group scheduling problem.

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

Contents

List of Figures

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 [4]. 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 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 [9]. 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 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 presents an extensive case study in which a transformation from a distributed agreement problem in a group scheduling problem is motivated. Finally, Section 5 concludes the paper.

2 Model Overview

The basic philosophy of the model is shown in Figure 1. We will first describe the model in general and will demonstrate it using 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.

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 $v_i, v_j \in V$.

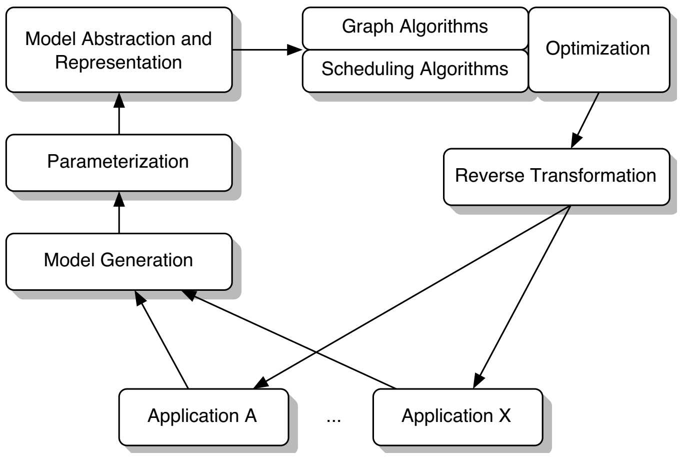

Figure 1: Model Overview

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 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 [**?**]. 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.

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 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 [3].

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.
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* [3]. 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 requirements 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 run 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 required 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 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 measure 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 processors are unrelated.

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 Study: Distributed Agreement

In this section we apply the model of Figure 1 to a problem that has been at the core of many applications in the area of fault-tolerance and survivability, i.e. fault-tolerant agreement. Reaching agreement in the presence of faults has been the subject of much research since the formulation of the "Byzantine General Problem" [15]. The agreement problem addresses value or event synchronization issues resulting from the use of redundancy as the preferred mechanism to allow for recovery from the affects of faults and malicious acts. In order to detect or mask any discrepancies among the redundant copies, voting mechanism of varying complexity must be in place. Agreement algorithms are the common solution. The algorithms considered range from simple majority voting to full Byzantine Agreement [15]. A wide pallet of agreement solutions exist, differing in the constraints put on the fault model and the networking environment. The main issues are (1) the fault model considered, e.g. benign, malicious, or mixed mode, (2) the network topology, e.g. fully or partially connected graphs, (3) the networking protocol, e.g. point-to-point or multicast, and (4) communication synchronization, e.g. synchronous or asynchronous.

One key issue is the time of the agreement, addressed by the notion of *immediate* and *eventual* Byzantine agreement [6]. Immediate agreement implies that all processors agree at the same time. In eventual agreement, all processors will eventually agree, but perhaps at different times.

The fault model and the agreement algorithms used in the replication scheme define the number of processors as well as the extent of the communication. Typically, as the number of processors increases, there is potential for dramatic increase in communication.

The demonstration of the model utilizes and relies heavily on portions of the research presented in [14], using the $OM(t)$ algorithm of Lamport [15] as a basis. The traditional $OM(t)$ agreement algorithm is a round-based recursive algorithm that requires $r = t + 1$ rounds of communication and a total of $N \geq 3t + 1$ processors in order to guarantee agreement in the presence of up to t malicious nodes.

4.1 Model Generation

The agreement algorithm can be mapped into an *agreement task graph* $G^A = (V^A, E^A)$. Task set V^A consists of two kinds of tasks, i.e. *real* and so-called *phantom* tasks. In [5] real tasks are defined as tasks in the usual sense, whereas phantom tasks are special tasks which consume time, but unlike real tasks, they consume no resources. Phantom tasks will be the primary mechanism for modeling communication. Therefore, we adjusted the definition in [5] slightly to reflect that a phantom task takes time but no CPU resources. This is motivated by a view of a network interface card as an autonomous identity that is not CPU bound. In the figures below, real tasks will be depicted by circles and phantom tasks by rectangles.

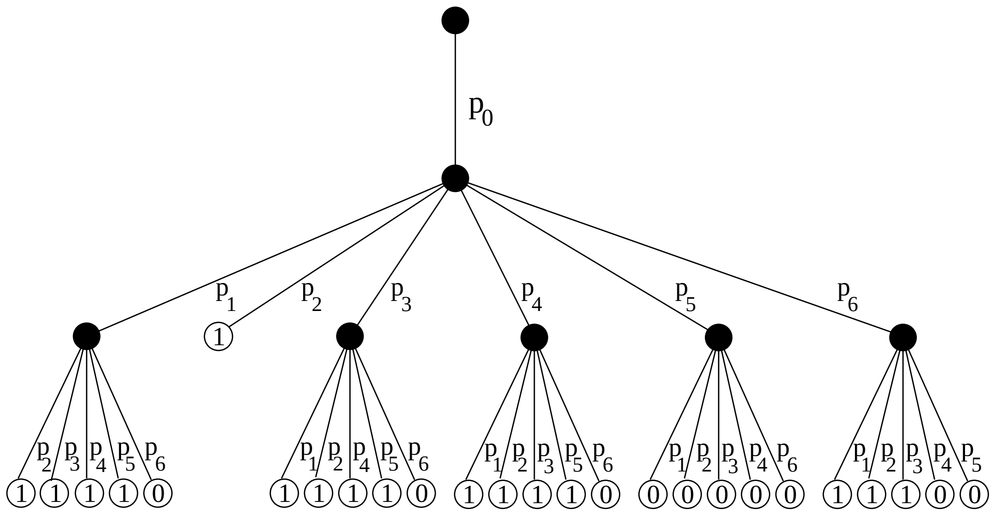

Figure 2: EIG-Tree of OM(2)

In the $OM(t)$ algorithm, each node keeps track of values received in a so-called EIG-Tree [2], where each level represents a round of message exchange. Figure 2 shows a sample EIG-Tree of processor P_2 for $OM(2)$. Each processing node maintains such a tree. Edges represent messages received, leaf vertices contain the received value. Edge labels indicate from which processor a value has been received. For example, following the path from the root to the right most leaf node indicates that this leaf node received a value that originated at processor P_0, was sent to P_6 and lastly received from P_5 with value 0. The depicted EIG-Tree has one single leaf node at level 2, all other leaf nodes are at level 3. This stems from the fact that the tree is stored at processor P_2, which does not need confirmation about its own value from others.

The EIG-Tree for the $OM(2)$ example of Figure 2 can be used to generate an agreement task graphs in a round-by-round fashion. With $t = 2$, a total of $r = t + 1 = 3$ rounds of communication are required. Figure 3 shows the task graph for round 0 and 1. Whereas the graphs are directed graphs, the arrows are omitted to avoid visual clutter. On the left side, the agreement *initiator* P_0 sends out messages, via phantom tasks, to all other processors. Each processor receives the messages and passes them on to the processor, which is indicated by the real task incident to the phantom task.

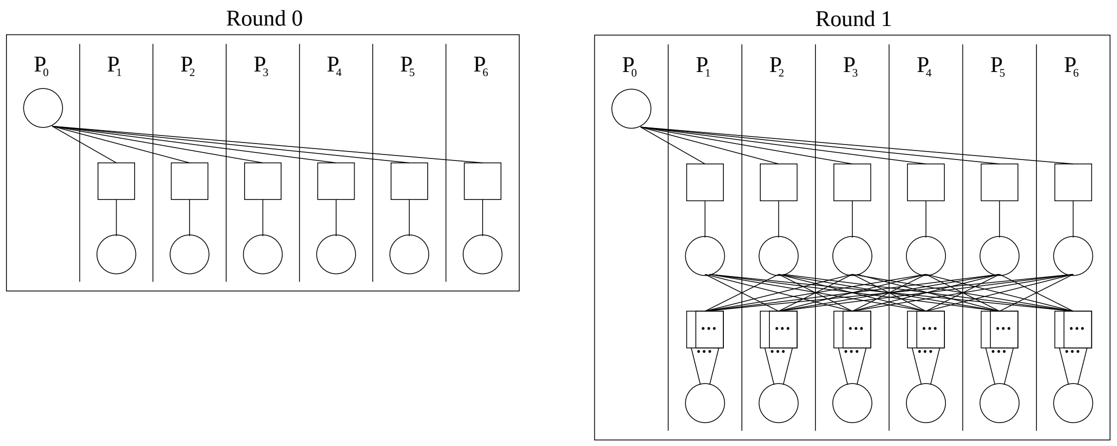

Figure 3: Task graph: $OM(2)$ round 0 and 1

In round 1 each processor forwards the value received from the initiator to all other processors. Each processor does so by acting as an "initiator" in the forwarding. Again, the message passing is modeled by phantom tasks, one per message sent. To enhance readability, this level of phantom tasks is represented in the figure as a group, with dots indicating the multiplicity. The leaf nodes indicate that all values have been received by the processor. The final task graph of $OM(2)$ can be seen in Figure 4 after the completion of the final round, i.e. round 2. It should be noted that, in the last round of communication in Figure 4, each edge represents $N - r$ messages. If one needs to consider each message separately, then each edge and the corresponding phantom task has to be replicated $N - r$ times. This is in consequence to the algorithm, which always sends each message received to all other processors that have not received it yet.

4.2 Parameterization

Now that the agreement task graph $G^A = (V^A, E^A)$ is available, e.g. Figure 4, the application specific parameters have to be introduced. 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 times p_i of each T_i needs to be specified. If T_i is a real task, then in a homogeneous processing environment 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*

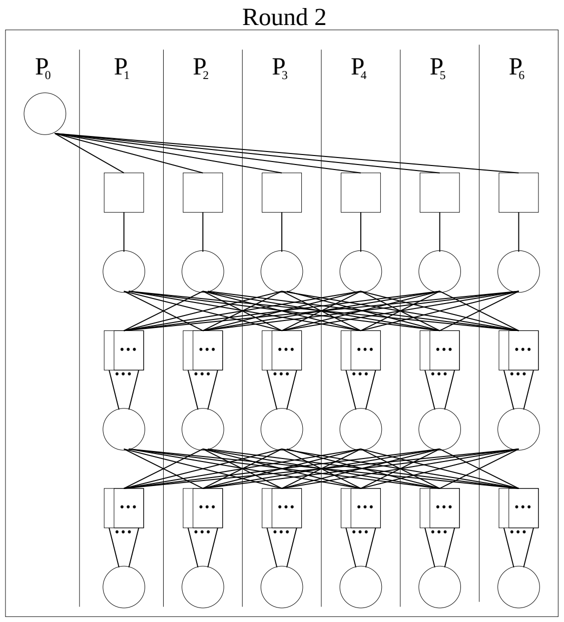

Figure 4: Task graph: $OM(2)$ round 2

value. Exact values for p_i can be either computation from the program segment or based on measurement. If T_i is a phantom task, than 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.

4.3 Model Abstraction and Representation

In [14] it has been shown that the performance of agreement algorithms is linked to how efficient certain task groups can be scheduled. Specifically, task primitives were defined that isolated the tasks at any specific level of the graph as a task group. Thus, with respect to Figure 4, task groups are defined by grouping together all tasks visited by each step of a breadth-first traversal of the graph, starting from the initiator task of P_0. It is the efficient scheduling of these task groups that determine largely the performance and degree of synchronization of the agreement algorithm.

Figure 5 illustrates the different times of interest during the process of reaching agreement. Assume that T_{init} is the initiating task of an agreement, e.g. the single task of P_0 in Figure 4. At time t^s_{init}, the initiating processor begins to send out messages to all other processors in order to start an agreement. At times $t^s_{p1}, t^s_{p2}, ..., t^s_{pn}$ the replicated tasks $T_{p1}, T_{p2}, ..., T_{pn}$, executing on different processors, receive the value from the initiating processor respectively. Each processor starts its message exchange process and tries to reach an agreement. At times $t^e_{p1}, t^e_{p2}, ..., t^e_{pn}$, the processors make a decision and reach agreement. Each processor then sends a signal back to the initiating processor to inform it that an agreement has been reached. The initiating processor receives the last message at t^e_{init}. From the initiating processor point of view, the whole agreement took Δt_{init} time units. However, it is possible that each processor will act immediately after it makes its decision, i.e. it does not need to report back to the initiating processor. Under this situation, the initiating processor does not know when exactly an agreement is reached. With respect to performance, the

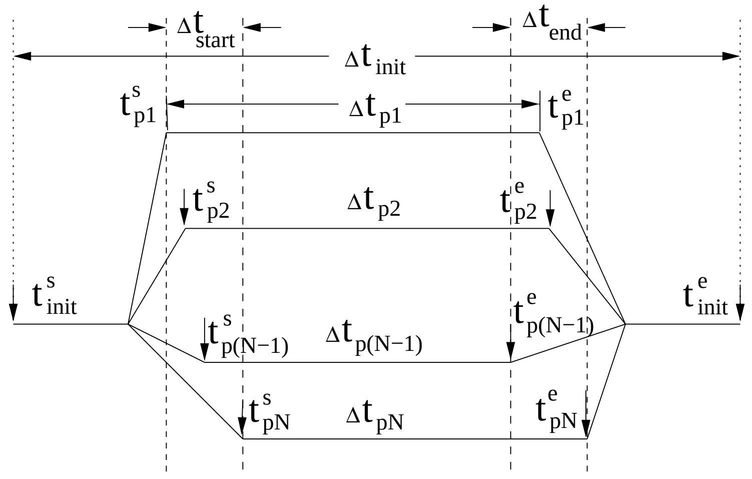

Figure 5: Communication timing

intervals Δt_{start} and Δt_{end} between the dashed lines are of special interest. Not all processors start and reach agreement at the same time. Interval Δt_{start} represents the maximum time difference among receiving processors when they become aware of the start of an agreement. If we assume that processors act immediately after they reach an agreement, then Δt_{end} indicates the maximum time difference among processors in taking action. This difference can be very important for many applications, since for Δt_{end} different processors can be in non-agreement states. If, for example, the agreement is in the context of database entries, then the distributed database is in an inconsistent state during this time.

The task groups of the agreement task graph $G^A = (V^A, E^A)$ must be scheduled on N

processors, together with the standard application tasks of each processor. For each P_i, $1 \leq i \leq N$, let $G_i^P = V_i^P, E_i^P$ be the processor specific static workload. 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 with identification of the task groups, as well as static task to processor assignments within a group.

4.4 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, however, it may be necessary to relax the C_{max} requirement. Thus, an optimal or suboptimal solution could be acceptable. It should be noted that it is not the scope of this research to identify the potential algorithms, we merely assume the existence of suitable solutions.

4.5 Reverse Transformation

The schedule derived with the algorithm selected in the previous step and the dispatcher satisfying the static allocation scheme need to be implemented. Thus, the reverse transformation in this application is simply the implementation of the scheduler and dispatcher.

Applying the model of Figure 1 to the distributed agreement case study resulted in a group scheduling problem formulation. We considered that the performance of the algorithm is directly linked to the level of synchronization of the task groups, i.e. it was affected by how narrow the intervals Δt_{start} and Δt_{end} of Figure 5 were. Theoretically, the optimal solution would be found by optimizing the schedule of the agreement task graph based on group scheduling with respect to the scheduling primitives defined in [14] for C_{max}. This was experimentally verified in [14].

5 Conclusion

This paper presented a model that can be used to find 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 a distributed agreement case study based on [14], which formulated the agreement problem as a group scheduling problem.

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] J. Allen, et. al., "State of the Practice of Intrusion Detection Technologies", Carnegie Mellon, SEI, Technical Report, CMU/SEI-99-TR-028, ESC-99-028, January 2000.

[2] 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.

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

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

[5] J. S. Deogun, R. M. Kieckhafer, and A. W. Krings, "Stability and Performance of List Scheduling With External Process Delays", *Real-Time Systems*, Vol. 15, No. 1, July 1998, 5-39.

[6] Dolev, D., et. al., "Eventual is Earlier than Immediate", 23^{rd} *Annual Symposium on Foundations of Computer Science*, pp. 196-385, 1982.

[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] E. Linger and M. Longstaff, *A Case Study in Survivable Network System Analysis*, Carnegie Mellon, SEI, Technical Report CMU/SEI-98-TR-014, 1998.

[9] 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.

[10] K. Calvin, et. al., *Detecting and Countering System Intrusions Using Software Wrappers*, Proc. 9th USENIX Security Symposium, 2000.

[11] Krings A.W, 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] Krings A.W, W.S. Harrison, 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, W. Harrison, 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, W. Harrison, M. Azadmanesh, and M. McQueen, *Scheduling Issues in Survivability Applications using Hybrid Fault Models*, to appear in Parallel Processing Letters.

[15] 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.

[16] P. Neumann, *Practical Architectures for Survivable Systems and Networks*, (Phase-Two Final Report), Computer Science Laboratory, SRI International, June 2000.

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