Optimal Monitoring Strategies for Critical Infrastructure Networks

Kartik Palani
Information Trust Institute
University of Illinois at Urbana-Champaign
palani2@illinois.edu

David M. Nicol
Information Trust Institute
University of Illinois at Urbana-Champaign
dmnicol@illinois.edu

ABSTRACT

Given a choice among multiple security solutions and multiple locations to deploy them, what strategy best protects the network? What metric is used to compare different securing strategies? What constraints make it harder/easier to secure critical infrastructure networks? This paper explores these questions and formalizes the network monitoring strategy problem for critical infrastructure networks. It also presents a deterministic polynomial time algorithm for discovering a near-optimal network monitoring strategy.

1. INTRODUCTION

The dependence of society on critical infrastructures such as energy, water, communications, and transportation make them targets for adversaries. The modernization of these infrastructures by integrating cyber components for better visibility and improved reliability has raised concerns of an increased attack surface. Attacks such as Stuxnet in Iran, BlackEnergy and CrashOverride in Ukraine, and Havex in the US have shown an increased sophistication and understanding that attackers have of critical processes to exploit the increased attack surface. The growth of cloud connected industrial internet of things devices in these infrastructures, and the interdependence among them leading to potential cascading and escalating failures further motivates the need for improved cybersecurity in these environments.

A key part of cyber defense in any system is monitoring network traffic and device behavior. However, there are key challenges in developing a good monitoring strategy. First, there are various types of monitors tasked with monitoring different behaviors and looking at different features. Choosing what should be monitored and where is a challenging task. Then, there are capital and operational costs to deploying monitors. The cost of a monitor is not just the cost of the hardware and installation cost but also the cost of analyzing the alerts (true and false positives) that they generate. Monitoring critical infrastructures has further constraints. There exist regulatory constraints, for example, in systems responsible for generation and transmission of sufficiently large volumes of electric power the NERC-CIP requirements apply, so that more devices means more work and expense in accounting for those devices in NERC-CIP audits. There also exist stringent timing constraints on the communications, violation of which could lead to instability of the physical processes in the system leading the system to an unsafe state. In fact, the NERC recommendation is that monitoring of sensors, logs, Intrusion Detection Systems (IDS), antivirus, patch management, policy management software, and other security mechanisms should be done on a real-time basis where feasible.

This work formalizes the network monitoring problem in critical infrastructures. *What is the best monitoring strategy in a given cyber network under the constraints stated above.* A monitoring strategy is a mapping of what flows must be monitored by what monitors deployed at what locations. We first define a metric for evaluating the goodness of a monitoring strategy, essentially to quantify the value of each mapping. The metric is based on the importance of the flows being monitored and where the monitor is relative to the endpoints of the flow. Choosing the best mapping from a combinatorially large space of mappings is NP-hard.

We describe an algorithm that can be used by network operators to make decisions, about near-optimal monitor deployments based on their security budget and the QoS requirements of the criticical infrastructure applications. The algorithm produces a valid assignment of flows to monitor types and locations, thereby allowing the administrator to make informed decisions about their network security strategy.

As a motivational example, consider the power grid transmission substation local area network (LAN) described in Figure 1. We specifically consider the application of synchronism checking to close a breaker. In order for stable and safe functioning of the power system, it is necessary that all electric loads and power generation be synchronized to a fixed frequency (60 Hz in the US). The synch check application ensures that before a breaker is closed the power systems on both sides of the breaker are in synchrony. At a high level, the application measures the angle between the single-phase voltage on either side (the bus and the line) and then determines if they are within set bounds. We focus on the communications that take place for this application to work:

- The substation host determines which sensors (voltage transformers) need to report the voltage and polls the corresponding intelligent electronic devices (IEDs).
- The VT IEDs add the time-stamp (gotten from a central time server) to the voltage and frequency information and send the message to the sync check relay.
- The sync check relay IED determines if the time-stamped voltages are in sync and issues a control command to

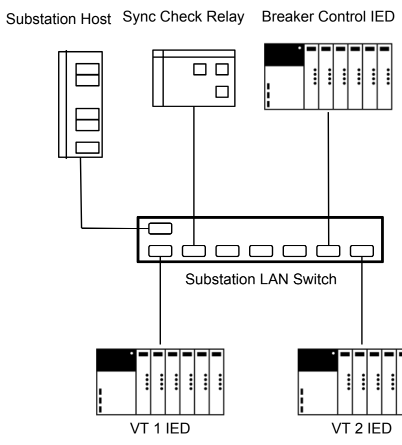

Figure 1: A single switch substation LAN

the breaker control IED.

- The breaker control IED interprets the control message and takes the corresponding control action.

The deadline requirement for the sync checking application is 100 microseconds from when a decision to close a breaker is made (this is what triggers the sync check application) to when the breaker is actually closed. Ideally, considering the criticality of breaker control, each of the messages would have to pass through a variety of security checks before each device acts on the message. Messages should be authenticated, detected for possible malformed packets, provenance checked, and payload must be inspected for malware. Which subsets of these functions can be applied? Which flows must be subject to these functions and which should pass uninterrupted? Note that in an actual substation multiple IEDs, hosts and switches exist and a transmission owner controls multiple substations that talk to each other and to a central control center over a wide area network.

2. PROBLEM STATEMENT

2.1 System Model

The cyber-network is modeled as an undirected graph $\mathcal{G} = (V, E)$ where V is the set of $|V|$ network elements: switches, routers, and hosts and where the set E of edges represents links between network elements.

A *monitor* is an appliance (deployed either as a middlebox, virtual network function or on the host) that can perform a range of security functions (firewall, intrusion detection/prevention system, authenticator, provenance verifier etc.) on flows it is assigned to monitor. Each monitor has a set M of security functions it supports. Monitors are placed on links, an assumption which admits for monitor implementation within devices at link endpoints, or placement of bump-in-the-wire monitoring devices on links. We suppose that only a subset of the links L can be monitored, $L \subseteq E$. If a monitor is deployed on link $l \in L$ then any flow passing through that link *might* be analyzed by the monitor, but as we will see, we do not necessarily assume that a flow passing through a monitor is analyzed, as the analysis imposes an additional latency cost we may need to avoid in order to meet real-time delivery constraints for that flow.

A *network monitoring strategy* is defined as an assignment *strat* of flows in F to the set $L \times M$ of link, security function pairs, such that a flow $f \in F$ is monitored by a networking function $m \in M$ at link $l \in L$. A *feasible* network monitoring strategy is one that meets all constraints (to be discussed at greater depth, shortly.)

Network traffic in critical infrastructures is often very well understood. The network engineers know which sources s communicate with which destinations t and they know the type of information being carried (e.g., sensor readings to the data aggregators, and system commands from the control station to system actuators.) They know the route that each flow traverses. It is reasonable to assume knowledge of a set F of traffic flows each of whose description is a tuple of (source device, destination device, source application), the last component being needed if multiple applications on a source device communicate with the destination device. Having recognized the distinction, for the remainder of the paper we leave off description of the source application, understanding that we can transform a problem where a source has multiple applications into a problem where each source has only one application, by creating virtual sources, one per application.

While, development of rational means of scoring the relative benefits of monitoring different flows using different monitors is work orthogonal to ours, we describe a formulation that allows for these scores to dictate the network monitoring strategy. Firstly, the value of monitoring commands to some actuators will be higher than others, depending on the importance of the actuator to the overall system, and the potential impact on the system of delivering falsified commands to it. Since not all flows are equally critical, we assume the existence of a function w which quantifies with a positive real number $w(f)$ the criticality of monitoring flow f.

Secondly, different security functions $m \in M$ have different impacts with respect to the flows being monitored, depending on the endpoints, the data the flow carries, and what precisely the monitor is able to detect. Consider flows carrying commands to actuators, if a security function can validate the integrity of the command then the monitoring may have a higher impact than if the function merely validates that the formatting of commands within the flow are valid with respect to the protocol specification. We will see that the problem formulation in this paper addresses the possibility of choosing one or many security functions from M such that the value of monitoring a particular flow with a particular type of security function depends on the attributes of the flow and the function. Let $\alpha_m(f)$ be the value of monitoring flow $f \in F$ with a security function $m \in M$.

The distance between the link at which the flow is monitored and its destination also adds affects the effectiveness of a monitoring strategy. Consider a flow f that is required to pass through an intrusion prevention system (IPS) and a data provenance monitor. In the case of the IPS it is beneficial to the bandwidth and security of the network if the flow is monitored closer to the network ingress point. Whereas for the data provenance check the value of monitoring closer to the destination is measurably better in order to detect data modifications within the network. Let $\gamma(d(l, f_t), m)$ be a function that quantifies the impact of applying security function m at link l which is $d(l, f_t)$ hops away from the destination f_t of flow f.

Finally, some security functions can benefit from monitoring a flow at multiple locations in the network. Consider an Intrusion Detection System for example, traditionally IDSs have been placed at the network ingress point, however, with growing concerns of attacks in the network data plane there have been efforts to also place IDS sensors internally. Other security functions such as authentication, however, only need to be applied once to a flow. Let β_m be an indicator of whether a monitor benefits from being re-applied ($\beta_m = 0$) or not ($\beta_m = 1$).

2.2 Monitor Performance Model

We assume that all monitors are alike in that they have the same set of security functions M and have similar processors. At run-time, each monitor will activate different combinations of security functions based on what flows are assigned to it.

Let $T \in \mathbb{R}^{p\times q}$ denote the timing matrix where $p = |F|$ and $q = |L \times M|$. An element T_{ij} is the worst case delay experienced by flow $i \in F$ at a security function $m \in M$ on link $l \in L$. The worst case delay is the time taken to process the flow considering all other flows that traverse the link l are assigned to the security function m. While this is a very conservative estimate, most of real time systems research takes this approach and considers worst case execution time as the metric for evaluating deadline constraints. Methods for computing worst case execution times are orthogonal to our work and are widely studied in real time systems research.

Each flow $f \in F$ has a deadline δ_f. There are certain flows that have flexible deadlines (defined in a min-max range) whereas most critical flows have strict upper bounds on the timing requirement as not adhering to this deadline might lead to system instability. We assume all deadlines to be strict i.e. δ_f is the absolute latest a flow can arrive at its destination. $\delta \in \mathbb{R}^{p\times 1}$ is the deadline vector where δ_i is the deadline of flow $i \in F$.

2.3 Network Monitoring Strategy Problem

Given the network, the flows in the network and the monitoring requirements of the flows, our goal is to find the network monitoring strategy that maximizes the monitoring quality obtained over the network while respecting the deadlines of the flows. Formally, we must determine a mapping *strat* from F to $L \times M$ that maximizes the monitoring benefit, $\mathcal{Q}(A)$. Here *strat* is a $|F| \times |L \times M|$ assignment matrix, where $strat_{ijk} = 1$ if flow i is assigned to be monitored using security function $k \in M$ placed on link j. For a valid strategy we must have that $T_f(l_1, m_1)x(l_1, m_1) + T_f(l_1, m_2)x(l_1, m_2) + ... + T_f(l_{|L|}, m_{|M|})x(l_{|L|}, m_{|M|}) \leq \delta_f$ for all $f \in F$, where $x(l_j, m_k)$ is an indicator of whether security function m_k on link l_j is chosen in the strategy. This defines the deadline constraint for a valid strategy. The metric for monitoring benefit, $\mathcal{Q}(A)$ is formally described as:

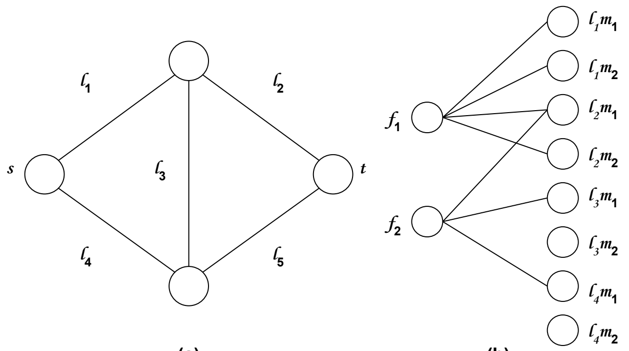

Figure 2: (a) shows the network graph $\mathcal{G}$. Consider two flows with paths $f_1 = l_1, l_2, f_2 = l_4, l_3, l_2$ and monitoring requirements $f_1 = m_1, m_2, f_2 = m_1$. (b) shows the bipartite graph, $\mathbb{B}$, that is constructed with edges that capture the routing information and monitoring requirement.

$$\mathcal{Q}(A) = \sum_{f\in A(F)} w(f)\Bigg[\sum_{m\in A_f(M)} \alpha_m(f)\Big[\beta_m\Big(\max_{l\in A_{fm}(L)} \gamma(d(l, f_t), m)\Big) + (1-\beta_m)\Big(\sum_{l\in A_{fm}(L)} \gamma(d(l, f_t), m)\Big)\Big]\Bigg] \tag{1}$$

where A is an assignment of flows to link-monitor pairs and $A(F)$ is the set of flows in the assignment, $A_f(M)$ is the set of security functions in A that are chosen for flow f, and $A_{fm}(L)$ is the set of links in A where flow f is monitored by security function m. Essentially, given an assignment, A, $\mathcal{Q}(A)$ returns the monitoring benefit of that monitoring strategy, A. Towards a formal understanding of the network monitoring strategy problem, consider the following optimization problem:

$$\underset{A\subseteq F\times L\times M}{\text{maximize}} \quad \mathcal{Q}(A) \quad \text{subject to} \quad Tx_A \leq \delta \tag{2}$$

An alternate graph-theoretic formulation of the problem is described in figure 2. Consider a bipartite graph $\mathbb{B} = (F \cup (L \times M), \mathbb{E})$. An edge $\mathbb{e} \in \mathbb{E}$ exists if the flow $f \in F$ passes through link $l \in L$ and requires monitoring using security function $m \in M$. The cost of selecting an edge $\mathbb{e}$ between flow f and link monitor pair lm is $c_{\mathbb{e}} = T_{flm}$ i.e. the worst case delay added to the flow f when it is monitored at link-monitor pair lm. The utility of choosing an edge is given by $\mathbb{Q}(\mathbb{e})$. An assignment A is a subset of edges in $\mathbb{E}$. Thus, the network monitoring strategy problem is one of choosing a valid subset $A \subseteq \mathbb{E}$ of edges in the bipartite, $\mathbb{B}$, such that the $\mathbb{Q}(A)$ is maximized. Note that an element $a_i \in F \times L \times M$ is the same as an edge in $\mathbb{E}$.

The problem described in equation 2 is NP-hard, as we will show later in this section. Our proposed algorithm will compute the assignments in a *greedy* fashion and we uncover and exploit the underlying *submodularity* structure of the problem to design and analyze the algorithm in a principled manner.

2.4 Properties of $\mathcal{Q}(A)$

The monitoring strategy scoring function in equation 1 has a number of key properties. By construction $\mathcal{Q}(A)$ is non-negative, as each of the constituent functions are non-negative. Furthermore, $\mathcal{Q}$ is normalized i.e. $\mathcal{Q}(\emptyset) = 0$. We also show that $\mathcal{Q}(A)$ has two more crucial properties: submodularity and monotonicity.

Definition 1. *Given a finite set X, a real-valued function f on the set of subsets of X is said to be **submodular** if it holds that:*

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B), \quad \forall A, B \subseteq X$$

*Alternatively, submodular functions can be defined through the property of **diminishing marginal values**:*

$$f(A \cup \{x\}) - f(A) \leq f(B \cup \{x\}) - f(B), \quad \forall B \subseteq A \subseteq X, x \in X \backslash A$$

The difference $f(A \cup \{x\}) - f(A)$ is known as the incremental marginal return of element x to set A and is denoted as $f_A(x)$.

The literature on approximation algorithms defines multiple problems where the goal is maximizing a monotone submodular function with multiple linear packaging constraints. These problems are of the form:

$$\max \quad f(S) \quad \text{s.t.} \quad Ax_S \leq b \quad \text{and} \quad S \subseteq [n]$$

where $A \in [0,1]^{m \times n}$, $b \in [1, \inf)^m$ and x_S is the characteristic vector of the set S. Each A_{ij} is the cost of including element j in the solution and each b_i is a budget constraint. The most common example of a problem that takes this form is the Knapsack problem (where the objective function is modular).

An additional property of interest is that of monotonicity:

Definition 2. *A function f is said to be **monotone** if for any $B \subseteq A$, $f(B) \leq f(A)$.*

To say that a function on sets is monotone non-decreasing is to say that if you take one argument set and increase it by adding one or more elements to the set, the function value never decreases as a result of the inclusion. It is clear that this is the case for $\mathcal{Q}$, since, adding a new link, network function, flow tuple to the assignment will either improve the current monitoring score or not affect it (if $w(f) = 0$ or $\alpha_m(f) = 0$ for all $m \in A(M)$).

With this background in submodularity, we can describe The network monitoring strategy discovery problem thus, falls in the general class of optimization problems

Theorem 2.1. *The function $Q : 2^{L \times M \times F} \rightarrow \mathbb{R}^+$ defined in equation 1 is submodular.*

We defer the proof of submodularity to the appendix.

Lemma 2.2. *The network monitoring strategy problem is NP-hard.*

Proof. The maximization of a monotone submodular function with linear packing constraints is NP-hard [2].

Having described some properties of the network monitoring strategy problem, we are ready to discuss a greedy algorithm that exploits these properties to achieves a constant factor approximation of the optimal.

Algorithm 1: Greedy monitor deployment algorithm

```
Input  : Worst case timing matrix $T$, deadlines $\delta$,
         update factor $\lambda$
Output: A set of link monitor pairs $A \subseteq L \times M$ with
         feasible deadlines
$A \leftarrow \{\}$ ;
$w_f \leftarrow 1/\delta_f$ for all $f \in F$ ;
$k \leftarrow 1$ ;
while $\sum_{f \in F} \delta_f w_f \leq \lambda$ and $A \neq L \times M$ do
    $j \leftarrow \text{argmax}_{j \in (L \times M) \backslash A} \frac{\mathbb{Q}_A(j)}{\sum_{f \in F} T_{fj} w_f}$;
    $A \leftarrow A \cup \{j\}$;
    $w_f \leftarrow w_f \lambda^{T_{fj}/\delta_f}$ for all $f \in F$;
    $k \leftarrow k + 1$;
    if $Tx_A \leq \delta$ then
        return $A$;
    else
        return $A \backslash \{j\}$);
    end
end
```

3. ALGORITHM

Our proposed solution for the network monitoring strategy problem is described in Algorithm 1. It takes as input the worst case timing matrix T, and the deadline vector δ and computes an assignment as the output. The algorithm starts with an empty assignment, *strat*, and proceeds in rounds (the while loop), where at every round a flow is assigned a link-monitor pair. The loop terminates when the assignment becomes such that the deadline of any flow is violated or when all flows have been fully monitored.

In order to address the multiple linear constraints, we must assign weights to each constraint and consider the linear combination of the costs as the true cost of every round. Note that each constraint can be written in the following form:

$$\sum_{\mathbb{e} \in \mathbb{E}} T_{f\mathbb{e}} \mathbb{1}_{\mathbb{e}} \leq \delta_f, \quad \forall f \in F \tag{3}$$

We initialize the weight of each constraint to be $w_f = 1/\delta_f$ for each flow $f \in F$ i.e. flows with the longest deadline are assigned a lesser weight and vice-versa. Informally, this weighting captures how close the constraint is to be violated under a given solution. The greedy step of the algorithm is such that it picks the $\mathbb{e}$ that maximizes the utility to cost ratio. The effective cost of selecting an edge in the bipartite graph, $\mathbb{B}$, is given by the normalized sum of weights, $\sum_{f \in F} T_{f\mathbb{e}} w_f$. The effective utility of choosing an edge $\mathbb{e} \in \mathbb{E}$ is given by $Q_A(\mathbb{e})$. Recall that this is the incremental marginal value of adding edge $\mathbb{e}$ to the assignment A. Thus, the ratio in equation 4 denotes the benefit of adding edge $\mathbb{e}$ per unit cost incurred. As we see in algorithm 1, we select an edge that maximizes this utility to cost ratio.

Once an edge has been selected in the first round, we must update the weights w_f on the constraints. The weight update step over all constraints (same as the number of flows) $f \in F$ is done as $w_f \leftarrow w_f \lambda^{T_{f\mathbb{e}}/\delta_f}$, where λ is an input parameter. Note that this weight change only affects flows in $A(f)$ and all other weights remain unchanged. The above process is repeated until $\sum_{f \in F} \delta_f w_f \leq \lambda$ is invalid. It can be shown that if a strategy returned violates any of the linear constraints (described in equation 3) then it must have happened in the last iteration of the loop and hence the last step of the algorithm is a constraint check which either returns the strategy *strat* or the strategy with the last chosen edge removed (which caused the violation).

It is only left to show that the edge $\mathbb{e}$ that maximizes

$$\frac{\mathbb{Q}_A(\mathbb{e})}{\sum_{f \in F} T_{f\mathbb{e}} w_f} \tag{4}$$

is found efficiently in each round (line 1 inside the while loop). Consider the edges from $\mathbb{E} \backslash A$. Adding an edge $\mathbb{e}$ to the assignment either adds a monitor of type $\beta_m = 0$ or $\beta_m = 1$. In case of the former, the function $\mathbb{Q}(A \cup \mathbb{e})$ is linear and the pre-computed weight of the edge can be added to $\mathbb{Q}(A)$ to get the new value of the assignment. In the case the monitor is of type $\beta_m = 1$ then max between the existing value of γ the value of $\gamma_{\mathbb{e}}$ is chosen. The graph distances between all link-destinations can be pre-computed using a breadth first search ($O(|F|(|V|+|E|))$). The computation of the utility cost ratio is then linear in the number of edges. A sorting operation over the computed ratios needs to be performed to get the edge that maximizes equation 4.

3.1 Complexity

The algorithm described is deterministic and efficient. In this section we explore the time complexity of the full algorithm and show that the complexity is at most $O(|\mathbb{E}|^2 log|\mathbb{E}| + |\mathbb{E}||F|)$. The single while loop will have at most $|\mathbb{E}|$ iterations. Now, in each iteration, it takes $O(\mathbb{E})$ operations to compute the utility of each edge and it takes another $O(\mathbb{E})$ operations to compute the utility cost ratios. The sorting of the updated utility cost ratios takes $O(|\mathbb{E}|log|\mathbb{E}|)$ operations. The weight update step takes $|F|$ iterations. Thus, each iteration of the loop takes $O(|\mathbb{E}|log||\mathbb{E} + |F|)$. Hence the complexity of the algorithm is $O(|\mathbb{E}|^2 log|\mathbb{E}| + |\mathbb{E}||F|)$.

In a bipartite graph, in the worst case $|\mathbb{E}| = |F \times L \times M|$. So the algorithm is almost linear in L and M and square in F.

3.2 Approximation Guarantee

We show that that algorithm 1 has a constant factor approximation guarantee. Let OPT and $ALG1$ denote the value of the objectives achieved by the optimal algorithm and algorithm 1 respectively.

We get an approximation guarantee of $(1-\epsilon)(1-1/e)$

Theorem 3.1. *Algorithm 1 approximates the optimal algorithm within a factor of* $(1-\epsilon)(1-1/e)$, *i.e.,* $\boldsymbol{ALG1} \geq (1-\epsilon)(1-1/e)\boldsymbol{OPT}$ *for some fixed* $\epsilon > 0$.

Let $\eta = min\{\delta_f / T_{f\mathbb{e}}\}$. Note that $T_{f\mathbb{e}} > 0$ holds since latency added at a link-monitor pair is always non-zero. η is the width of the packing constraints. Theorem 1.1 in [1] states that $ALG1$ attains an approximation guarantee of $\Omega(1/|F|^{1/\eta})$ and this is a constant factor approximation since the number of constraints, $|F|$, is constant for a given matrix T. Note that the update factor λ is essential in the approximation guarantee of interest. In the general setting, an update factor of $\lambda = e^{\eta}|F|$ (Lemma 2.4 in [1]) can be used to get the constant factor approximation, $\Omega(1/|F|^{1/\eta})$. The special case approximation factor described in Theorem 3.1 holds when an update factor of $\lambda = e^{\epsilon\eta/4}$ is employed in $ALG1$ when the instance of the problem is such that $\eta = \Omega(ln|F|/\epsilon^2)$ for any fixed $\epsilon > 0$. This holds when the timing matrices are dense with small skew.

4. DISCUSSION

Note that our current problem formulation allows for adding a cost constraint. This could, for example, be used to describe monetary cost of monitoring a flow. A new row in the matrix, T, can be added where each element is the cost c_i of monitoring edge $\mathbb{e}$ i.e. the money spent in monitoring a flow $\mathbb{e}_f$ using security function $\mathbb{e}_m$ on link $\mathbb{e}_l$, say due to an analyst that must study the alerts generated. The constraint vector δ would have a corresponding budget below which the total cost must be.

Similarly, if there is a restriction on the total number of link-monitor pairs that must be chosen (to reduce the number of appliances deployed) then a row in T such that $T_{1j} = 1$ can be added and the quantity constraint is added to δ.

We note the worst case timing matrix makes very strict assumptions about the effect of latency on a flow, thus making the network monitoring strategy produced by our methodology more time conservative in the latency vs security tradeoff. This potential for leeway in the timing matrix, due to the fact that flows have schedules and priorities, motivates future work.

5. REFERENCES

[1] Y. Azar and I. Gamzu. Efficient submodular function maximization under linear packing constraints. In *International Colloquium on Automata, Languages, and Programming*, pages 38–50. Springer, 2012.

[2] A. Krause and D. Golovin. Submodular function maximization.

Appendix

Proof of Theorem 2.1

Consider two assignments $A \subseteq A' \subseteq F \times L \times M$. We have by definition of $\mathbb{Q}(A)$ that when a new element $(f_0, l_0, m_0) \setminus A$ is added to the assignment

$$\mathbb{Q}(A \cup (f_0, l_0, m_0)) = \mathbb{Q}(A \setminus A_{f0}(m_0)) +$$
$$w(f_0)\Big[\alpha_{m0}\Big[\beta_{m0}[max_{l \in A_{m0f0}(l) \cup l_0}\gamma(d(l, f_t), m)] +$$
$$(1-\beta_{m0})[\sum_{l \in A_{m0f0}(l) \cup l_0} \gamma(d(l, f_t), m)]]]$$

Adding a new edge only affects the monitoring score for flow f_0 when monitored by type m_0 i.e. $A_{f0}(m_0)$. If the monitor is of type $\beta_{m0} = 0$ then there is a linear increase in score and if it is of type $\beta_{m0} = 1$ then there is either no change in score or an increment. To prove submodularity we must show that $\mathbb{Q}_A(f_0, l_0, m_0) = \mathbb{Q}(A \cup (f_0, l_0, m_0)) - Q(A) \geq \mathbb{Q}_{A'}(f_0, l_0, m_0)$. In the case that $\beta_{m0} = 0$ the effect of adding a new tuple is $w(f_0)\alpha_{m0}(f_0)\gamma(d(l_0, f_0t), m_0)$. In this trivial case, the function is linear (modular) since the effect of adding the tuple is equal to both assignments. The more interesting case is when $\beta_{m0} = 1$. When this type of monitor exists, there can be three cases: *Case 1:* the new link maximizes γ for neither A nor A'. In this case,

$\mathbb{Q}_{A'}(f_0, l_0, m_0) = 0 = \mathbb{Q}_A(f_0, l_0, m_0)$ making the function modular. *Case 2:* the new link maximizes γ for A but not A'. In this case $\mathbb{Q}_{A'}(f_0, l_0, m_0) = 0$ but $\mathbb{Q}_A(f_0, l_0, m_0) > 0$, thus making the function submodular. *Case 3:* the new link maximizes γ for both assignments. In this case $\mathbb{Q}(A' \cup (f_0, l_0, m_0)) = \mathbb{Q}(A \cup (f_0, l_0, m_0)) > 0$ but we know from monotonicity that $\mathbb{Q}(A') \geq \mathbb{Q}(A)$ thus,

$$\mathbb{Q}(A \cup (f_0, l_0, m_0)) - \mathbb{Q}(A) \geq \mathbb{Q}(A' \cup (f_0, l_0, m_0)) - \mathbb{Q}(A') \blacksquare$$