Security and Robustness Issues in Collaborative Runtime Verification

Bas Testerink[1], Nils Bulling[2], and Mehdi Dastani[1]

[1]Utrecht University, [2]Delft University,
{B.J.G.Testerink,M.M.Dastani}@uu.nl,{N.Bulling}@tudelft.nl

Abstract. Decentralized monitors can have robustness and security risks. Among robustness risks are attacks on the monitor's infrastructure in order to disable parts of its functionality. Among security risks are attacks that try to extract information from the monitor, and thereby possibly leak sensitive information. Formal methods to analyze these issues given a monitor design can help to make better designs and/or identify critical parts in a monitor. In this paper we specify a model for analyzing robustness and security risks for monitors where a network of runtime local monitors forms a collaborative monitor.

Keywords: Monitoring, Runtime Verification, Security

1 Introduction

Normative systems help to make sure that agents behave according to preset guidelines/norms in multi-agent systems [5]. One approach are exogenous normative systems where norms are explicit. The normative aspect of the multi-agent system is captured by an exogenous - to the agents - organization or institution. With this approach it must be verified whether any norm violations occur in the multi-agent system's behavior. Monitoring large distributed multi-agent systems such as traffic, smart grids and economic markets requires decentralized approaches. Monolithic centralized monitors can impose a bottleneck due to the distributed nature of multi-agent systems and a single point of failure in case of break downs.

A major concern of decentralized verification techniques is their robustness and security. The data that is gathered from a multi-agent system can severely compromise the agents' privacy if leaked. Adversaries can also try to take down parts of the networks to impede its functioning. Formal models of decentralized monitors can identify the critical parts in monitors in terms of robustness and security. This helps in their design and implementation as extra resources can be invested in critical parts. In this paper we present an abstract model for decentralized verification monitors and investigate the properties of robustness and security issues that have to be faced when designing a monitor. As an example setting we shall use a traffic monitoring scenario (Example 1). Traffic monitoring faces many challenges, including physical attacks on the monitor infrastructure and the privacy of individuals (cf. [7]).

Monitors in our approach observe the execution trace of a system to verify properties of its behavior. These properties are expressed in linear-time temporal logic (LTL) formulas [13]. LTL has been used in the past to analyze normative systems (e.g. [1]). Hence we shall focus on LTL verification, with the knowledge that various normative frameworks rely on LTL. For the structure of decentralized monitors we draw inspiration from existing popular decentralized monitoring techniques such as wireless sensor networks (WSNs). The structure of a WSN is that a collection of information gathering nodes routes sensor data towards a so-called sink. Intermediate data aggregation is often used to increase security and save energy. In our decentralized verification framework a network of local monitors collaborates to verify properties. Hence we call such a network a collaborative monitor. Each local monitor has observation capabilities of its own and can query other local monitors to verify other properties. This is similar to the frameworks proposed in [3] and [16]. However, different from those two frameworks we assume that information flows through the network without their models of information delay. Also, instead of sharing observations as in [16] or passing around progressed formulas as in [3], local monitors in our framework combine their input into a Boolean evaluation and share that with whichever other local monitors need it. This represents the aggregation of input.

Our contribution is a formal framework for specifying collaborative monitors with which we analyze the critical robustness and security parts within them. We formally evaluate the framework. It is our belief that the presented framework is not only beneficial to the design and development of decentralized monitors for LTL verification, but can also provide insight on design-time and/or runtime analysis of robustness and security risks for other decentralized verification technologies.

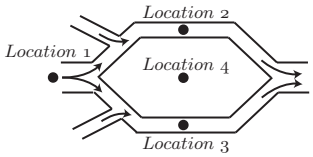

Fig. 1. Example scenario. Black dots indicate locations, arrows indicate traffic flow and double lines indicate roads.

Example 1 (Scenario). We will discuss an example traffic situation (Figure 1). In this scenario there is a road side unit (RSU) that is a notification board about traffic jams. There are two routes to the same end location which may be jammed. A local monitor at location 1 observes whether cars that pass the RSU end up in a traffic jam in spite of the provided information. On both routes (location 2 and 3) there are local monitors with short-range communication capabilities that can check for a specific car whether it is using that route, and also whether the route is jammed. This information is aggregated by a local monitor with long-range communication capabilities at location 4 and sent to the local monitor at location 1. This scenario is global through all examples.

The rest of the paper is structured as follows: In section 2 we summarize related work and background literature that is of influence in this paper. In

section 3 we present the model for collaborative monitors that we use to describe robustness in section 4 and security in section 5. In section 6 we discuss possible changes in the framework's assumptions and future work.

2 Related Work and Background

The field of wireless sensor networks contains a vast amount of identified robustness and security risks and countermeasures (cf. [11], [10]). The aim is to keep the monitoring service online if local monitors malfunction and prevent sensitive information from being obtained by an adversary. The requirements of WSNs are commonly organized by: 1) data confidentiality (only intended receivers can see sensitive data), 2) data integrity and freshness (data is correct and new), 3) protection against Sybil attacks (the imitation of monitors), 4) availability (continued operation of monitors).

Protection against confidentiality and availability attacks will be the main focus of this paper. Data integrity and freshness is assumed. Different kinds of attacks can be categorized between attacks that change the network topology (e.g. physically compromising a node or communication line, or a wormhole attack that connects two nodes) and those that extract information from the network (capture and/or imitation of nodes). We shall address the case that a local monitor can malfunction. This encapsulates both aggressive and non-aggressive failures of local monitors. We shall also discuss the case that an attacker can query a local monitor without proper authorization. This encapsulates Sybil attacks.

On top of being vulnerable to attacks WSNs also suffer from hardware constraints. Sensors tend to have limited power and communication capabilities. A common practice to limit energy usage is to use intermediate aggregation of data between the source of data and the sink. Data aggregation also helps to optimize any other monitoring system by reducing the amount of required communication. There are many works dedicated to security in data aggregation systems [9]. Do note that in data aggregation literature privacy is also often intended as the privacy of which sensor provided what contribution to the aggregated information. In this paper we consider only privacy in the form of the privacy of agents. Aggregation in our framework manifests itself by having local monitors combine their input into a single evaluation which is shared with other local monitors.

Security related papers on wireless networks for monitoring tend to focus on how to prevent security risks by using cryptography (e.g. [12]) and/or special routing protocols (e.g. [8]). Our approach is complementary to this. We do not look at runtime implementation techniques for preventing risks, but address design time questions and analysis to see where potential robustness and security risks lie. We believe design based analysis can help in further improving decentralized monitors. Depending on the practical limitations of an application it might not be possible to always make a perfect design. But our work can help in determining which parts of a network require more advanced/expensive hardware to increase safety.

In contrast to WSNs, we shall take a logical approach as this fits better with the declarative nature of normative systems theory. Many normative systems express norms as conditional obligations/prohibitions with deadlines (cf. [2]). Such constructions can often be expressed as properties about a system's behavior over time. This has led us to opting for linear temporal logic. The runtime verification process of LTL can be modeled with automata [4] or progression systems [6]. For our system it is not important which one is used. As for decentralized LTL verification there are the systems from [3] and [16]. However the framework in [3] is built on assumptions that do not support our intended scenarios. For one, in their framework all monitors are connected to each other whereas we want to investigate specific topologies. The framework from [16] does allow different topologies but data is not aggregated by local monitors. Also we have no notion of information delay, which both frameworks have.

As in [3] and [16] we assume that the monitors work synchronously. In various decentralized monitoring communication protocols synchronization is introduced in order to have data freshness (e.g. SNEP [12]). We assume that the monitors are connected in an acyclic manner and that the aggregation operation of local monitors is taking a Boolean combination of the input.

3 Monitor Model

The task of a monitor is to verify whether the monitored system's behavior satisfies some property. The behavior of the system is a trace of states, and properties are expressed as LTL formulas. We first provide the preliminaries. We then proceed to explain the architecture of collaborative monitors and how views of local monitors and their in-between connections give rise to their locally verifiable properties. A preliminary version of our verification model is to be published in [15].

3.1 Formal Setting

Let Π be a set of propositional symbols. A transition system (over Π) is given by a tuple $T = (S, R, V)$ where S is a state space, $R \subseteq S^2$ a serial transition relation, and $V : S \to 2^{\Pi}$ an evaluation function which returns the propositions that hold for a given state.

An *infinite trace* is defined as an infinite sequence $\sigma = s_0 s_1 \dots$ of states interconnected by R, i.e. $(s_i, s_{i+1}) \in R$ for all $i \in \mathbb{N}_0$. Similarly, a *finite trace* is given by $s_0 \dots s_k$. The set of infinite and finite traces is denoted by Tr_T^i and Tr_T^f, respectively. The set of all traces is denoted by $\mathsf{Tr}_T = \mathsf{Tr}_T^i \cup \mathsf{Tr}_T^f$. The *length* of a trace, i.e. the number of states on it, is denoted by $|\sigma|$; in particular, if $\sigma \in \mathsf{Tr}_T^i$ then $|\sigma| = \infty$. We use $\sigma[i]$ to refer to state i on σ where $0 \leq i < |\sigma|$. Also, we will often take the traces as first-class citizen if it is not important by which transition system it was generated, and omit mentioning T as subscript if it is clear from the context. Given a finite trace σ we write $\sigma\mathsf{Tr}^i$ to refer to all infinite

traces in Tr^i that have σ as initial prefix. We take Π, S, V and Tr to be global throughout this article.

We shall use an unaltered version of linear-time temporal logic LTL from [13]. Formulas of LTL are defined by the following grammar:

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \psi \mid \bigcirc\varphi \mid \varphi \mathcal{U} \psi, \text{ where } p \in \Pi.$$

As per usual we use $\Box\varphi$ (always φ) as a syntactical abbreviation for $\neg(\top \mathcal{U} \neg\varphi)$ and $\Diamond\varphi$ for $\neg\Box\neg\varphi$ (somewhere in the future φ).

Definition 1 (Infinite Trace Semantics). *Let $T = (S, R, V)$ be a transition system, $\sigma \in \mathsf{Tr}^i$ be an infinite trace and $i \in \mathbb{N}_0$ an index. The infinite trace semantics for* LTL *are defined by relation $\models$ as follows:*

$$\begin{aligned}
T, \sigma, i &\models p && \Leftrightarrow p \in V(\sigma[i]) \\
T, \sigma, i &\models \neg\varphi && \Leftrightarrow T, \sigma, i \not\models \varphi \\
T, \sigma, i &\models \varphi \vee \psi && \Leftrightarrow T, \sigma, i \models \varphi \text{ or } T, \sigma, i \models \psi \\
T, \sigma, i &\models \bigcirc\varphi && \Leftrightarrow T, \sigma, i+1 \models \varphi \\
T, \sigma, i &\models \varphi \mathcal{U} \psi && \Leftrightarrow \exists j \in [i, \infty] : T, \sigma, j \models \psi \text{ and} \\
& && \quad \forall k \in [i, j-1] : T, \sigma, k \models \varphi
\end{aligned}$$

Finite trace semantics for LTL evaluate a formula to t, f or ? (unknown). The intuitive reading for t is that the property holds for a given finite trace, therefore also for all finite and infinite extensions of that trace. Analogously f means that the property does not hold. And ? means that in the current finite trace can be extended s.t. the property holds or does not hold.

Definition 2 (Finite LTL Semantics). *Let $T = (S, R, V)$ be a transition system, $\sigma \in \mathsf{Tr}^f$ be a finite trace and $i \in [0, |\sigma| - 1]$ an index. The finite trace semantics for* LTL *are defined by relation $[\cdot]$ as follows:*

$$[\varphi]^T_{\sigma,i} = \begin{cases} t & \text{if } \forall \sigma' \in \sigma\mathsf{Tr}^i : \sigma', i \models \varphi \\ f & \text{if } \forall \sigma' \in \sigma\mathsf{Tr}^i : \sigma', i \not\models \varphi \\ ? & \text{otherwise} \end{cases}$$

We write $[\varphi]^T_\sigma$ for $[\varphi]^T_{\sigma,0}$, Moreover, we use $\sigma, i \models^3 \varphi$ to refer to $[\varphi]^T_{\sigma,i} = t$, and $\mathsf{Tr}^f \models^3 \varphi$ if for all finite traces $\sigma \in \mathsf{Tr}^f$ we have that $\sigma, 0 \models^3 \varphi$.

3.2 Local and Collaborative Monitors

Each local monitor has its own sense capabilities which allows the local monitor to verify an LTL formula on any finite execution trace. Local monitors also receive some input evaluations from neighbors. A monitor aggregates the evaluation of its observation formula with all the input evaluations using a Boolean function.

Definition 3 (Local Monitor). *A local monitor $m = (f, \varphi)$, where f is a Boolean function over k Boolean variables ($k \geq 1$) called the* aggregation function*, and φ is an LTL formula, called the* observation formula*.*

A collaborative monitor is modeled by a directed acyclic graph of local monitors. The main reason for acyclicity is to avoid unnecessary complexities for the formulation of collaborative LTL verification. The connections between local monitors is referred to as the query relation. This relation is given by a function that given a local monitor m returns the connected local monitors in order of their input for the aggregation function of m.

Definition 4 (Collaborative Monitor Specification). *A* collaborative monitor C *is specified by* (M, qry)*, where* M *is a non-empty set of local monitors,* $\mathsf{qry} : M \to \bigcup_{k=0}^{\infty}(M^k)$ *is a query relation with* $\mathsf{qry}(m) = (m_1, \ldots, m_k)$ *iff* $m = (f, \varphi)$ *where* f *is a Boolean function over* $k+1$ *inputs. Moreover,* qry *is assumed to be acyclic*[1]*.*

Example 2 (Collaborative monitor specification). Figure 2 shows a representation of the example scenario. The RSU is located at location 1. For simplicity we will talk about a specific car in the examples. Local monitors m_1, m_2 and m_3 can observe the location of the car (l_i stands for "the car is at location i"). m_2 and m_3 can also observe traffic jams at their location (j_i means "location i is jammed"). In the traces we consider traffic flows from l_1 to l_2 and l_3, and each place will be visited maximally once in the trace. m_4 aggregates whether the car has been in a traffic jam. Formally the collaborative monitor from the example scenario is specified by (M, qry) s.t.:

- $M = \{m_1, m_2, m_3, m_4\}$.
- $\mathsf{qry}(m_1) = (m_4)$, $\mathsf{qry}(m_2) = \mathsf{qry}(m_3) = ()$ (no input), $\mathsf{qry}(m_4) = (m_2, m_3)$.

And the local monitors (o is the observation formula's evaluation, x and y are input evaluations):

- $m_1 = (f_1, \varphi_1)$, $f_1(o, x) = o \wedge x$, $\varphi_1 = \Diamond l_1$.
- $m_2 = (f_2, \varphi_2)$, $f_2(o) = o$, $\varphi_2 = \Box(\neg j_2 \vee \neg l_2)$.
- $m_3 = (f_3, \varphi_3)$, $f_3(o) = o$, $\varphi_3 = \Box(\neg j_3 \vee \neg l_3)$.
- $m_4 = (f_4, \varphi_4)$, $f_4(o, x, y) = x \wedge y$, $\varphi_4 = \top$.

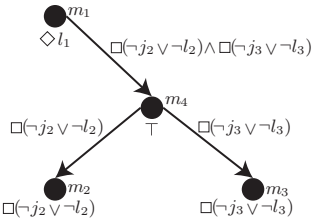

Fig. 2. Example collaborative monitor. Nodes are local monitors with at the right their name, arrows indicate the query relation, below monitors are the observable formulas.

[1] Let *reachable* be inductively defined as: m' is reachable from m if m' is among $\mathsf{qry}(m) = (m_1, \ldots, m_k)$. Furthermore by transitivity if m'' is reachable by m' and m' is reachable from m, then m'' is reachable by m. Acyclicity means that there is no $m \in M$ s.t. m is reachable from m.

Given a local monitor m in a collaborative monitor C the resulting LTL formula that represents how m aggregates its input evaluations is called m's aggregate. If a local monitor m can query m', then it means that m can query the evaluation of the aggregate of m'. Because for each local monitor the aggregation function is fixed, it means that given a collaborative monitor all the aggregates can be determined before runtime. Also note that due to acyclicity there are local monitors $m = (f, \varphi)$ without neighbors and hence for such m's their aggregate is equivalent to φ or $\neg\varphi$.

Definition 5 (Aggregate, Q_m). *Let T be a transition system, $C = (M, \mathsf{qry})$ be a collaborative monitor, and $m = (f, \varphi) \in M$ be a monitor. m's aggregate is a given LTL formula Q_m s.t. for each $\sigma \in \mathsf{Tr}^f$:*

$[\mathsf{Q}_m]^T_\sigma \equiv f([\varphi]^T_\sigma, v_1, \ldots, v_k)$, *where* $\mathsf{qry}(m) = (m_1, \ldots, m_k)$, $v_i = [\mathsf{Q}_{m_i}]^T_\sigma$ *and* Q_m *is a Boolean combination of* φ *and* $\mathsf{Q}_{m_1} \ldots \mathsf{Q}_{m_k}$.

Example 3 (Aggregate). The aggregate Q_{m_1} is equal to $\varphi = \Diamond l_1 \wedge \Box(\neg j_2 \vee \neg l_2) \wedge \Box(\neg j_3 \vee \neg l_3)$. The reading of φ is that the car passes somewhere in the future l_1 and never in the future ends up in a traffic jam.

Local monitors operate in a synchronous manner. This means that at runtime, starting by monitors without neighbors, the aggregate evaluations are computed until for each monitor the aggregate is evaluated, before the system makes another transition. If in practice the system is continuous, then each system transition can be seen as a new sample. In those cases all aggregate evaluations are computed before the next sample is taken.

3.3 Monitorability and Expressivity

For various robustness and security issues, and from a design perspective, it is useful to determine what kind of formulas can be collaboratively verified by some local monitors. We first define the notion of (non)monitorability, which is an adaptation of the one from [14] and [4]. For a formula φ being nonmonitorable after a finite trace σ means that given finite LTL semantics, φ can never be evaluated to a conclusive true or false after σ. For φ to be monitorable it means that there is no trace s.t. φ is nonmonitorable after that trace.

Definition 6 ((Non)monitorable). *Let $C = (M, \mathsf{qry})$ be a collaborative monitor, $m \in M$ be a local monitor, $\sigma \in \mathsf{Tr}^f$ be a finite trace, and φ be an arbitrary* LTL*-formula. We say that φ is* Tr*-nonmonitorable after σ iff for all finite traces $\sigma' \in \mathsf{Tr}^f$ s.t. σ is a prefix of σ': $[\varphi]^T_{\sigma'} =?$. We say that φ is* Tr*-monitorable iff there is no finite trace $\sigma \in \mathsf{Tr}^f$ s.t. φ is* Tr*-nonmonitorable after σ by m in C.*

Remark 1 (Complexity Monitorability). In general, checking whether a formula φ is monitorable is a difficult task, it is at least PSPACE*-complete, independent of whether a set of traces is given or not. One of the hard problems is the complexity of* LTL *model and satisfiability checking (over finite traces), which is a* PSPACE*-complete problem.*

We are interested in whether a local monitor m can verify a specific LTL formula φ for any trace. The aggregate of m is syntactically defined. Therefore, the condition is that φ is equal to Q_m semantically. This would guarantee that for any finite trace the evaluation of Q_m is equal to the evaluation of φ. We say that φ is finitely observable if not only Q_m is equal to φ but φ is also monitorable.

Definition 7 ((Finitely) Observable). *Let $C = (M, \mathsf{qry})$ be a collaborative monitor, $m \in M$ be a monitor, and φ be an arbitrary LTL-formula. We say that φ is* Tr-observable *by m in C iff $\mathsf{Tr}^f \models \mathsf{Q}_m \leftrightarrow \varphi$.*

We say that φ is finitely Tr-observable *by m in C iff φ is Tr-observable by m and φ is Tr-monitorable.*

Example 4 (Monitorability and observability). An example formula that is non-monitorable after any finite trace is $\Box\Diamond(j_2 \vee j_3)$ (there will always be a jam in the future). The formula $\psi = \Diamond l_1 \wedge \Box((\neg j_2 \vee \neg l_2) \wedge (\neg j_3 \vee \neg l_3))$ is equivalent to Q_{m_1} and hence Tr-observable by m_1. Furthermore, because ψ is Tr-monitorable, it is also Tr-finitely observable by m_1.

Remark 2 (Useless monitors). If φ is Tr-nonmonitorable then, as intuition would demand, it might be useless to build a monitor to observe this property. E.g. for $\varphi = \Box\Diamond p$ this is the case, because it is Tr-nonmonitorable after ϵ (empty trace). Any local monitor m in any C would not be able to finitely Tr-observe φ, and m will only be able to evaluate φ to unknown. The same holds for a specific transition system T' iff there is a shared prefix σ for all traces in $\mathsf{Tr}^f_{T'}$ s.t. φ is $\mathsf{Tr}_{T'}$-nonmonitorable after σ.

We assume that each collaborative monitor has the purpose that one or more local monitors can verify a specific formula. The target formulas for local monitors are captured by expressiveness constraints.

Definition 8 (Expressiveness Constraints). *Expressiveness constraints for a collaborative monitor $C = (M, \mathsf{qry})$ are defined as $E \subseteq M \times \mathsf{LTL}$ which are pairs of local monitors with the formulas that must be Tr-observable for them. C Tr-satisfies the expressiveness constraints E iff for each $(m, \varphi) \in E$: φ is Tr-observable by m in C.*

Example 5 (Expressiveness Constraint). An expressiveness constraint for the example scenario is $E = \{(m_1, \varphi)\}$ where φ is from example 2. Because φ is Tr-observable by m_1 it means that the example collaborative monitor satisfies its expressiveness constraints E.

4 Robustness

Suppose we are given a collaborative monitor which satisfies some expressiveness constraints. For various reasons, among which are physical sabotaging attacks, local monitors and/or communication links between them can malfunction. From a system designer's perspective it can make sense to construct a monitor with

some redundancy such that the expressiveness constraints are still satisfied when some components malfunction. In this section we analyze the robustness of monitors; that is, to which degree local monitor failures affect the functioning of the collaborative monitor.

4.1 Monitor Malfunctioning

If a local monitor fails then it is removed from the collaborative monitor. For a local monitor it may mean that one of its inputs does not contribute to the aggregate anymore.

Definition 9 (Local monitor malfunctioning). *Let $C = (M, \mathsf{qry})$ be a collaborative monitor, and $F \subseteq M$ be a set of malfunctioning monitors. We define the collaborative monitor $C|_F = (M', \mathsf{qry}')$ s.t.:*

- $M' = M \setminus F$.
- $\mathsf{qry}'(m)$ *equals* $\mathsf{qry}(m)$ *with the absence of local monitors in* F.
- *Each* $m = (f, \varphi) \in M \setminus F$ *becomes* $m = (f', \varphi)$ *s.t. every* $m' \in F$ *is removed from the input for* f' *and each occurrence of* $(\neg)\mathsf{Q}_{m'}$ *in* Q_m *is replaced by* $\bot$ *in* f'[2].

Example 6 (Local monitor Malfunctioning). If $F = \{m_4\}$ then $C|_F = (M', \mathsf{qry}')$ s.t. $M' = \{m_1, m_2, m_3\}$, and $\mathsf{qry}'(m_i), i \in \{1, 2, 3\}$, is the empty sequence. Note that now $Q_{m'_i}$ is equal to the observation formula of m_i. For $m_1 = (f'_1, \varphi_1)$ in $C|_F$ the aggregate Q'_{m_1} becomes Q_{m_1} (which without $\wedge$ is $\neg(\neg\varphi_1 \vee \neg\mathsf{Q}_{m_4})$) where the occurrences $\neg\mathsf{Q}_{m_4}$ and Q_{m_4} are replaced by $\bot$ (i.e. $\mathsf{Q}'_{m_1} = \neg(\neg\varphi_1 \vee \bot) = \varphi_1$).

We limit ourselves to local monitor malfunctioning, but communication malfunctioning can be straightforwardly defined as well. Instead of removing local monitors, the query relation is updated. If communication from m_1 to m_2 malfunctions then the new query relation removes m_2 from m_1's input sequence of local monitors. The aggregation function is updated the same as with local monitor malfunctioning. An extension of malfunctioning where monitors send false information is also possible. One could model this by altering the aggregation function.

4.2 Monitor Robustness

In a hostile environment local monitors can be damaged, but they can also malfunction for other reasons (e.g. running out of energy). We aim at quantifying robustness in terms of how much damage a collaborative monitor can take before its expressiveness constraints are not satisfied any more. This damage can be expressed as a set of potentially malfunctioning local monitors or a number of malfunctioning local monitors.

[2] For this definition it is required that $\wedge$ is not used as a shorthand operator.

Definition 10 (Collaborative monitor robustness). *Let $C = (M, \mathsf{qry})$ be a collaborative monitor satisfying the expressiveness constraints E, $M' = \{m | m \in M \wedge (m, \varphi) \notin E\}$, $F \subseteq M$ a subset of local monitors, and $k \in \mathbb{N}_0$. We say that C is F-robust for E and Tr if $C|_F$ Tr-satisfies E. We say that C is k-robust for E if C is F'-robust for E for any $F' \subseteq M'$ with $|F'| \leq k$.*

We note that given a collaborative monitor C 0-robustness is equivalent to $\emptyset$-robustness for some E and Tr which simply means that C Tr-satisfies E. If every local monitor occurs in E with some φ then only 0-robustness can be obtained.

Example 7 (Collaborative monitor robustness). Let $F = \{m_4\}$ and E be as in example 5. The example collaborative monitor is not F-robust for E. However, if we imagine that m_2 and m_3 can potentially switch to more energy costly but long range communication, then the monitor would be F-robust. In that case $\mathsf{qry}(m_1)$ would include m_2 and m_3 so that Q_{m_1} in C is equivalent to $\mathsf{Q}_{m_1'}$ in $C|_F$. Also, given E the monitor can only be 0-robust.

Given a practical scenario there might be potential failures that are very likely to occur. For those potential failures it is important to see whether the design of the monitor is robust.

Proposition 1. *Let $C = (M, \mathsf{qry})$ be a collaborative monitor, E be expressiveness constraints, $M' = \{m | m \in M \wedge (m, \varphi) \notin E\}$, $F \subseteq M'$, and T a finite transition system. The problem of checking F-robustness for E and Tr_T is in the same complexity class as determining Tr-observability.*

$C|_F$ can be constructed in polynomial time. Once $C|_F$ is constructed the question can be answered by checking for every $(m, \varphi) \in E$ whether the representing monitor m' can Tr-observe φ.

Aside from a specific attack we might wonder how much damage a monitor can take in general before it fails. This is especially useful in scenarios with many homogeneous local monitors such as botnets where attacks can be widespread and targeting any point in the network. The higher k-robustness that can be achieved, the more robust the monitor is.

Corollary 1. *Let $C = (M, \mathsf{qry})$ be a collaborative monitor, E be expressiveness constraints, $k \in \mathbb{N}_0$, and T a finite transition system. The problem of checking k-robustness for E and Tr_T is in the same complexity class as determining Tr-observability.*

It is combinatorial to check for all subsets F of M of size k whether C is F-robust for E and Tr_T. Each of these checks involves $|E|$ amount of Tr-observability checks.

4.3 Fail Tolerance

We set out to provide a framework for determining critical parts of a collaborative monitor. For data availability this is determined by robustness. Intuitively

we want to capture for a local monitor how critical its functioning is for the collaborative monitor. We shall use a fairly simple qualification called fail tolerance for determining how critical a local monitor is. This can be used as a basis for more sophisticated metrics. Recall that the presented analysis is for designs. In an implementation one has to deploy a mechanism for detecting whether a monitor has failed.

For a collaborative monitor C with expressiveness constraints E, being k-fail tolerant for $m \in M$ can be read as: k is the smallest size for a set F including m s.t. C is not F-robust for E and for any subset F' of F C is F'-robust for E. If $k = \infty$ then it means that for any set F where m is included s.t. C is not F-robust for E there is a strict subset $F' \subset F$ where $m \notin F'$ and C is not F'-robust for E.

Being 1-fail tolerant for m would mean that m's functioning is absolutely critical for the collaborative monitor, i.e. if m fails then the collaborative monitor does not satisfy its expressiveness constraints. Being k-fail tolerant for m, $1 < k < \infty$, would indicate that there is some redundancy to m's functioning. Being ∞-fail tolerant would indicate that the collaborative monitor does not depend on m's functioning for satisfying its expressiveness constraints.

Definition 11 (Fail Tolerant). *Let $C = (M, \mathsf{qry})$ be a collaborative monitor and E be its expressiveness constraints. For a monitor $m \in M$ we say m is k-fail tolerant, $k < \infty$ iff there is a $F \subseteq M$ of size k s.t.:*

- *(1) $m \in F$, C is not F-robust for E, for each subset $F' \subseteq F \setminus \{m\}$ C is F'-robust for E, and*
- *(2) there is no $F' \subseteq M$ s.t. (1) holds for F' and $|F'| < |F|$.*

Otherwise m is ∞-fail tolerant.

Determining the fail tolerance of a local monitor is also a difficult task as checking for F-robustness is involved. However, for applications where the expressed properties for expressiveness constraints and/or local monitor observations are not LTL it can be less hard.

Example 8 (Fail tolerance). All the monitors in our example scenario are 1-fail tolerant, as each of them observes vital information for the goal aggregate in m_1. However, see figure 3 for illustrations of more robust collaborative monitors. In the left monitor we assume only one expressiveness constraint s.t. m_7 must be able to aggregate $p \wedge q \wedge r$, hence m_7 is 1-fail tolerant. m_1 is 1-fail tolerant as its failure will immediately let the whole monitor fail. m_2 is 2-fail tolerant as its failure together with m_3 will let the whole monitor fail. Note that m_1 together with m_2 is not considered for m_2's fail tolerance as m_1 was already 1-fail tolerant. Monitors m_4 to m_6 are 3-fail tolerant.

In the right collaborative monitor we assume that m_5 must be able to aggregate $p \wedge q$ and hence it is 1-fail tolerant. All the other monitors are 2-fail tolerant, and the collaborative monitor as a whole is 1-robust.

If a monitor is k-robust, then the lowest i-fail tolerance that a local monitor can have aside from 1 is $(k + 1)$.

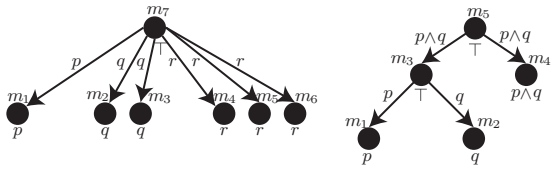

Fig. 3. Example collaborative monitors. Nodes are local monitors with on the top their name, arrows indicate the query relation, below monitors are the observable formulas.

Proposition 2. *Let $C = (M, \mathsf{qry})$ be a collaborative monitor, E be its expressiveness constraints and $M' = \{m | (m, \varphi) \notin E\}$. C is k-robust iff all monitors $m \in M'$ are at least $(k+1)$-fail tolerant.*

Proof sketch: If C is k-robust for E then there are no sets $F \subseteq M'$ where $|F| \leq k$ s.t. C is not F-robust for E. This implies that for any subset $F' \subseteq M$ where $|F| \leq k$, if C is not F'-robust for E then some $m \in M \setminus M'$ is also in F', and hence for a subset of F' C is also not robust. Thus for any monitor $m \in M'$ the minimum size subset m can be member of s.t. there no smaller subset for which the monitor is not robust, must be $(k+1)$. Hence each $m \in M'$ is at least $(k+1)$-fail tolerant.

Given a design we can now see how critical a local monitor is in terms of its availability. A straightforward expansion of this fail tolerance analysis is to not only check for F-robustness, but also look at which expressiveness pairs (m, φ) are not satisfied anymore in case the monitor is not F-robust. If not all expressiveness constraints are equally important, then for tolerance analysis this can be taken into account. Also looking at how many pairs are not satisfied is an important ingredient should one want to specify graceful degradation for collaborative monitors. At runtime the collaborative monitor can benefit from fail tolerance analysis by assigning higher reparation priorities to critical monitors, if possible.

5 Security

When it comes to the security of a collaborative monitor, we focus on the possible information an attacker can extract from the monitor. We say an attacker extracts an LTL property φ if the attacker can extract from the collaborative monitor the evaluation of φ given an arbitrary finite trace.

5.1 Information Extraction

In practice there are various ways in which an attacker can extract information. The attacker can intercept and interpret messages, query other monitors by pretending to be authorized and capture a monitor. To analyze intercepted messages it is required to know what messages are exactly exchanged, something which is not covered in our model. Also our model is not suited for analyzing

how a monitor can be reprogrammed, given that we only have semantical representations and no program specifications of monitors. Therefore we focus on the situation where the attacker can ask a query by pretending to be authorized. This can for instance in practice take form if the attacker pretends to be a local monitor from the network.

We assume that like monitors the attacker can aggregate extracted information by using some Boolean function. However, for the attacker this is not a fixed function. Hence, given the aggregates that it obtains from the collaborative monitor it can combine these in any Boolean manner (notated with PL).

Definition 12 (Monitor attack). *Let $C = (M, \mathsf{qry})$ be a collaborative monitor. An attack $\mathsf{att} \subseteq M$ is a set of local monitors. The set of extracted properties $\mathcal{L}^C_{\mathsf{att}}$ is $\mathsf{PL}(\{\mathsf{Q}_m | m \in \mathsf{att}\})$.*

If the framework is expanded and other forms of attacks can also be analyzed then these will contribute to the set of extracted properties. Note that if some φ is extracted from the monitor, then the attacker may still not know whether the system's behavior satisfies this property given a finite trace. Because the evaluation may be inconclusive.

Example 9 (Monitor attack). If an attacker can imitate m_4 then it will be able to query m_2 and m_3. In that case $\mathsf{att} = \{m_2, m_3\}$ and the extracted properties are $\mathsf{PL}(\{\mathsf{Q}_{m_2}, \mathsf{Q}_{m_3}\})$. So for instance the attacker can determine given an arbitrary trace what the evaluation is of $\neg\Box(\neg j_2 \vee \neg l_2) \vee \neg\Box(\neg j_3 \vee \neg l_3)$ (somewhere in the future there was a jam at either location and/or the car was at either location).

5.2 Safety

For monitor safety we look at whether a specific attack extracted a specific property from the monitor. We assume the attacker knows what an aggregate represents. I.e. if it obtains information on the evaluation of m's aggregate then it knows that m's aggregate is Q_m.

Definition 13 (Monitor Safety). *Let C be a collaborative monitor, att be an attack and φ be an LTL formula. We say that C is safe for φ and att iff there is no $\psi \in \mathcal{L}^C_{\mathsf{att}}$ s.t. $\mathsf{Tr}^f \models^3 \varphi \leftrightarrow \psi$.*

Example 10 (Monitor Safety). Given $\mathsf{att} = \{m_1, m_2, m_3, m_4\}$ the example collaborative monitor is safe for $\varphi = \Diamond(l_1 \wedge \Diamond l_2)$ and att. It is also safe for $\psi = \Diamond(l_1 \wedge \Diamond l_3)$ and att. This means that even if the attacker can obtain all available aggregates, it still cannot determine for a trace whether the car used or may use in the future the route through locations 1 and 2 or through locations 1 and 3.

The space of potential attacks is heavily restricted by practical details that are not covered by our model. For instance if in a network some local monitor only has wired connections to other local monitors in a safe environment, then it might be impossible that that local monitor is targeted for an attack. Therefore we focus

on analyzing security risks w.r.t. potentially attacked local monitors and with the assumption that the attack is practically feasible. The security constraints of a monitor consist of a set of monitors that can potentially be attacked and a set of properties that represent sensitive information. The analysis of what properties count as sensitive should be part of the system's design methodology. These will differ per practical real-world scenario. A monitor satisfies its security constraints if none of the considered attacks allows the attacker to monitor a sensitive property.

Definition 14 (Security Constraints). *Security constraints for a collaborative monitor $C = (M, \mathsf{qry})$ are defined as (A, P) where $A \subseteq M$ is a set of local monitors and $P \subseteq \mathsf{LTL}$ is a set of sensitive properties. C satisfies its security constraints iff for each $\mathsf{att} \in 2^A$ and $\varphi \in P$, C is safe for φ and att.*

It should be noted that for some constraints (A, P) a nonmonitorable property $\varphi \in P$ might be nonsense to consider as sensitive information, just like tautological formulas. If φ for instance can only evaluate to "?" then the attacker does not need to extract the property to know φ's evaluation for any finite trace.

Example 11 (Security Constraints). Let $M = \{m_1, m_2, m_3, m_4\}$, φ and ψ be as in example 10 and $(A, P) = (M, \{\varphi, \psi\})$ be the scenario's security constraints. The example collaborative monitor satisfies its security constraints (A, P). This means that no matter what local monitors are attacked, the route of the car remains private.

Remark 3. Like monitorability, the problem of checking whether some collaborative monitor C satisfies its security constraints is also a difficult task that is at least PSPACE-complete. Even if an attack contains a single monitor, then checking equivalence with a sensitive property is equal to LTL equivalence which is PSPACE-complete.

5.3 Attack Tolerance

For security we also want to know how critical a monitor is. I.e. if a certain monitor's aggregate can be obtained, how bad is that? In contrast to robustness, this is only interesting if the collaborative monitor *does not* satisfy its security constraints. Because that would indicate that there are combinations of monitors s.t. the attack on those monitors would reveal sensitive information.

Like our approach for fail tolerance, we shall use a simple basis for attack tolerance that can be expanded into more sophisticated methods. The attack tolerance of a local monitor can be read as the distance that the attacker is from obtaining sensitive information if the monitor is attacked. The intuition is that a local monitor is the less tolerant (and more critical) the less other local monitors need to be attacked before sensitive information is extracted. A local monitor is maximally attack tolerant if it cannot contribute to any security leakage at all.

Definition 15 (Attack tolerant). *Let $C = (M, \mathsf{qry})$ be a collaborative monitor and (A, P) be its security constraints. For an attack $\mathsf{att} \subseteq A$, a local monitor $m \in \mathsf{att}$, and $\mathsf{att}' = \mathsf{att} \setminus \{m\}$: we say m is contributing to att iff $P \cap \mathcal{L}^C_{\mathsf{att}} \neq \emptyset$ and $P \cap \mathcal{L}^C_{\mathsf{att}} \neq P \cap \mathcal{L}^C_{\mathsf{att}'}$.*

For a monitor $m \in M$ its k-attack tolerance is defined as:

- *m is ∞-attack tolerant iff there is no $\mathsf{att} \subseteq A$ s.t. $m \in \mathsf{att}$ and m is contributing to att.*
- *m is i-attack tolerant iff $\mathsf{att} \subseteq A$ is the smallest attack s.t. m is contributing to att and $i = |\mathsf{att}|$.*

If $C = (M, \mathsf{qry})$ satisfies its security constraints then all local monitors are ∞-attack tolerant. If a local monitor is 1-attack tolerant then its aggregate is equivalent to a sensitive property, or the aggregate's negation is. The maximal attack tolerance value aside from ∞ is $|M|$.

Example 12 (Attack tolerant). In our example scenario the monitor satisfies its security constraints and hence all monitors are ∞-attack tolerant. If in in the lower monitor from figure 3 the security constraints are $(\{m_1, \ldots, m_7\}, \{p \wedge q\})$ then m_1 and m_2 are 2-attack tolerant and the other local monitors are 1-attack tolerant.

Now we can determine how attack tolerant a local monitor is, which is an indicator for how critical the local monitor is for security. From here on there can be expansions that make a more detailed analysis of the severity of attacks and/or take into consideration how hard it is to attack a specific monitor. For instance if one monitor is easier to attack than another (i.e. a simple sensor in a WSN versus a sophisticated sink), then the tolerance of the easier target should be decreased relatively to the harder target. In a runtime environment if attacks are detected then new attack tolerance values can be computed with the knowledge that some local monitors are already attacked. This could for instance be countered with low attack tolerance local monitors switching to more secure, albeit energy and/or more overhead cost expensive, communication protocols.

6 Discussion & Conclusion

In this paper we presented a formal framework for collaborative monitors, which are networks of local monitors. With this framework we then defined how we can determine the critical parts of the network. This and future contributions will help to improve monitor designs and multi-agent system robustness and security. For our formal setting we used LTL as the target language to specify system behavior properties. This can be changed to other languages. Of course many definitions would change but the cornerstone concepts of our framework; local and collaborative monitors, robustness, safety and fail/attack tolerance still remain intact. The same holds for changing the aggregation function to more or less expressive aggregation.

Throughout the paper we explained at important points where the framework could be expanded. We want to continue investigating robustness and security for decentralized monitoring by continuing to analyze more precisely communication failures and communication protocols at runtime between local monitors in a collaborative monitor.

References

1. T. Agotnes, W. Van Der Hoek, J. Rodriguez-Aguilar, C. Sierra, and M. Wooldridge. On the logic of normative systems. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1181–1186. 2007.
2. N. Alechina, M. Dastani, and B. Logan. Reasoning about normative update. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 20–26. AAAI Press, 2013.
3. A. Bauer and Y. Falcone. Decentralised LTL monitoring. In D. Giannakopoulou and D. Méry, editors, *FM 2012: Formal Methods*, volume 7436 of *Lecture Notes in Computer Science*, pages 85–100. Springer Berlin Heidelberg, 2012.
4. A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, Sept. 2011.
5. G. Boella and L. V. D. Torre. Introduction to normative multiagent systems. *Computational and Mathematical Organization Theory*, 12:71–79, 2006.
6. K. Havelund and G. Rosu. Monitoring programs using rewriting. In *Proceedings of the 16th Annual International Conference on Automated Software Engineering(ASE 2001)*, pages 135–143. IEEE, 2001.
7. B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. Enhancing security and privacy in traffic-monitoring systems. *IEEE Pervasive Computing*, 5(4):38–46, Oct. 2006.
8. C. Karlof and D. Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. *Elsevier's AdHoc Networks Journal, Special Issue on Sensor Network Applications and Protocols*, 1(2–3):293–315, September 2003.
9. S. Ozdemir and Y. Xiao. Secure data aggregation in wireless sensor networks: A comprehensive overview. *Comput. Netw.*, 53(12):2022–2037, Aug. 2009.
10. A. Pathan, H.-W. Lee, and C. S. Hong. Security in wireless sensor networks: issues and challenges. In *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*, volume 2, pages 6 pp.–1048, Feb 2006.
11. A. Perrig, J. Stankovic, and D. Wagner. Security in wireless sensor networks. *COMMUNICATIONS OF THE ACM*, 47(6):53–57, 2004.
12. A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. SPINS: Security protocols for sensor networks. *Wirel. Netw.*, 8(5):521–534, Sept. 2002.
13. A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57, Oct 1977.
14. A. Pnueli and A. Zaks. PSL Model Checking and Run-Time Verification Via Testers. In J. Misra, T. Nipkow, and E. Sekerinski, editors, *FM 2006: Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 573–586. Springer Berlin Heidelberg, 2006.
15. B. Testerink, N. Bulling, and M. Dastani. A model for collaborative runtime verification. In *Proc. of the 14th Int. Conf. on Autonomous Agents and Multiagent Systems (t.b.p.)*, AAMAS '15.
16. B. Testerink, M. Dastani, and J.-J. Meyer. Norm monitoring through observation sharing. In A. Herzig and E. Lorini, editors, *Proceedings of the European Conference on Social Intelligence*, pages 291–304, 2014.