Monitoring Security and Dependability in Mobile P2P Systems

George Spanoudakis, Department of Computing, City University, London

Kelly Androutsopoulos, Department of Computing, City University, London

Abstract

Ensuring the dependability and security of mobile P2P systems is an intricate task due to the autonomous and decentralised nature of such systems. In this paper, we present a framework that provides increased support for security and dependability properties by monitoring the compliance of the operation of mobile P2P applications with them at runtime. The framework performs monitoring driven by policies specified for the individual peers in a P2P application and decouples the monitoring process from the operation of the application, to increase its resilience and avoid adverse effects on its performance.

Keywords

Runtime security and dependability monitoring, mobile P2P systems

1 INTRODUCTION

The proliferation of advanced wireless access infrastructures, and advances in hardware miniaturisation, enable software-based Internet applications of increasing complexity to find their way into mobile devices such as mobile phones and PDAs. Equipped with such applications, a mobile device can become a business tool, allowing its user to perform various complex tasks whilst on move. And as high-speed ad-hoc connectivity becomes readily available, mobile software applications are becoming a significant driving force behind the adoption of a mobile phone as a mobile computing platform.

In this realm, peer-to-peer software applications − i.e., applications in which there is no distinction between clients and servers and every participating entity can play the role of a *client*, a *server* and a *router* at the same time − become important due to their inherent capability of distributing content and computations over a network of mobile devices. Furthermore, P2P architectures enable higher scalability due to their decentralised structure [1]. Thus, several mobile P2P applications have emerged recently including, P2P wireless positioning [31], mobile P2P file sharing [32] and enterprise instant messaging applications [21, 30].

Despite the benefits which arise from the decentralised and dynamic nature of mobile P2P applications, however, this very nature is also posing some significant challenges for security and dependability. To appreciate these challenges consider the case of enterprise P2P instant messaging applications. Typically such applications provide platforms for building communities with common financial interests that can use P2P messaging to exchange information about financial markets, trading news etc. Monitoring the *availability* of the peers that provide the authentication service to a P2P messaging network at runtime, for instance, is important in order to avoid the presence of unauthorised nodes in the network. Monitoring is also important for detecting *denial of service* attacks. Such attacks may be caused by flooding the authorisation peers with network admission requests making them and, consequently, the whole service unavailable to legitimate peers. Also, giving individual peers the capability to restrict or totally prevent interactions with other peers, which are not considered trustworthy, is important for their own *integrity* and confidence in the P2P service.

Runtime checks may, of course, be implemented by the P2P application itself but this approach is not flexible enough to cope with the evolution of the P2P network and changes in the requirements of individual peers which may arise due to this evolution. Thus, relying on built-in checks is likely to require costly extensions in the implementation of the P2P application. An alternative solution to this problem is to monitor security and dependability requirements of individual peers at runtime using external monitors. Runtime monitoring has been proposed as a technique that complements static verification and testing and several approaches have been developed to support it including [2, 9, 11, 16, 27]. Typically, a runtime monitoring framework provides mechanisms for capturing events during the operation of a system and checking whether the captured events satisfy specific properties. Most of the existing runtime monitoring frameworks, however, focus on computing platforms where resource scarcity is not a significant constraint, systems which are not as dynamic and decentralised as mobile P2P applications or on monitoring network communications rather than an application level operations To this end, they do not address adequately some essential issues for monitoring and controlling mobile P2P systems, notably:

- the need to have a monitoring service that is not deployed on the same machine as the peers that it monitors, in order not to drain the computational and power resources of mobile devices;
- the need to support dynamic negotiations between mobile peers at runtime in order to enable the activation of monitoring activities on them;
- the secure emission of events from mobile P2P systems required for monitoring; and
- the dynamic execution of actions to prevent or rectify detected violations of the monitored properties during the execution of peer applications.

In this paper, we present a mobile peer verification framework (referred to as *MPVF* henceforth), which we have developed to provide the above monitoring and control capabilities for mobile P2P systems. MPVF's operation is driven by *monitoring policies* which are specified for individual peers to express *application level properties* that should be monitored at runtime and *control actions* that should be taken when the properties are violated. A peer monitoring policy specifies properties that should be monitored on the peer itself and/or other peers interacting with it within a P2P network. A policy also specifies *permissions* that a peer is willing to give to its collaborators to enable them to monitor and control its own activities. Furthermore, MPVF supports the negotiation and activation of monitoring policies across peers, the collection of events which are required for monitoring from individual peers, the transmission of these events to monitors and the application of control actions when violations of specific properties are detected. MPVF is part of a runtime platform that has been developed to secure the operations of mobile P2P systems as part of the EU research project PEPERS. In addition to the monitoring capabilities of MPVF, this platform provides support for identity management, data communication confidentiality, peer authentication and access control (see [14] for a full account of these features which are beyond the scope of this paper).

In the rest of this paper we focus on MPVF. More specifically, in Section 2 we present the architecture of MPVF; in Section 3 we introduce the language for specifying MPVF policies; in Section 4 we discuss policy activation and control; in Section 5 we discuss related work; and, finally, in Section 6 we provide conclusions and plans for future work.

2 Architecture of MPVF

Architecturally, MPVF has two key characteristics. The first characteristic is that it decouples monitoring from event capturing and control giving individual peers responsibility for the latter two activities and an external monitor responsibility for the former activity. The second characteristic is that it deploys a publish/subscribe notification infrastructure in order to transmit monitoring events from individual peers to the external monitors and monitoring results in the opposite direction. Figure 1 shows the overall architecture of MPVF which consists of three types of components, namely *monitoring-enabled peers* (MEPs), *monitors* and *event brokers* (EBr).

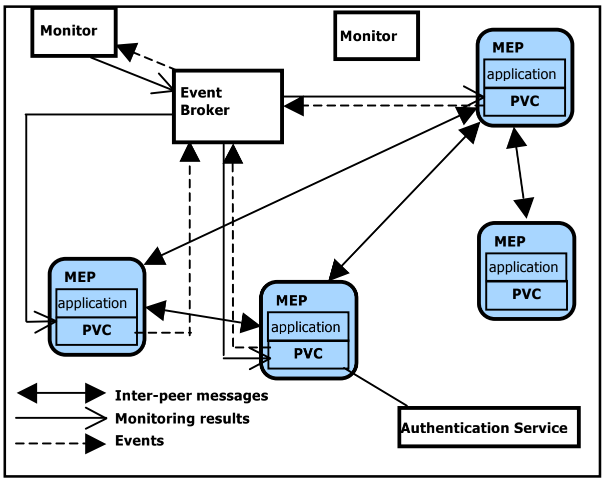

Figure 1. Architecture of MPVF

Monitoring enabled peers are peers that incorporate a *peer verification controller* (PVC). PVC collects events during the operation of the peer and publishes them to event brokers, so that they can be distributed to appropriate monitors. It also has responsibility for receiving notifications of results of the monitoring process and taking control actions on the individual peer as required by these results (e.g., dropping messages that exchanged between a peer and its collaborators before they reach their destination). As shown in Figure 2, PVC is provided as part of the basic runtime infrastructure that enables the formation of peer networks (e.g. peer registration, authentication and discovery) and the communication between the individual peers in these networks. Thus, when a peer application is built using this infrastructure, it automatically incorporates PVC. PVC internally consists of a *controller*, a *policy parser* and a *negotiation manager*.

The PVC controller intercepts all the incoming and outgoing messages, which are exchanged between its host peer and other peers, and publishes these messages in the form of encrypted events to the event broker of the MPVF, if necessary. The messages that need to be published to event brokers are determined by the *monitoring policy* that has been defined for the specific host peer and agreements that this peer may have made with other peers for exposing events that would enable their monitoring activities. After sending a message to an event broker, the controller may block it until it receives a notification that the message does not violate any rule or permit its transmission, and wait for the asynchronous notification of monitoring results. Subsequently, when it receives the monitoring results that relate to the message, the PVC controller applies the actions required by the active monitoring policies.

The *policy parser* of PVC is responsible for parsing the monitoring policy of the host peer and creating a repository with information about the types of events that should be intercepted, the properties that should be checked for these events and the actions that should be taken if properties are violated. Finally, the *negotiation manager* of PVC enables it to negotiate with external peers and agree or not the receipt and exposition of events that are necessary for checking the active monitoring policies at each side. The negotiation process is driven by the monitoring policies of the involved peers. PVC comes pre-assembled with the peer communication infrastructure and signed off so that it cannot be circumvented or tampered with.

The monitor in MPVF is a reasoning engine that checks whether the rules in a monitoring policy are satisfied by the events which are generated by the peers at runtime. A monitor may undertake monitoring tasks on behalf of several peers depending on its operational capacity and MPVF may use more than one monitor, each having responsibility for different nodes in a peer network. In this way, MPVF can provide more efficient and resilient to failure monitoring. When a peer appoints a monitor M, it subscribes M to the event broker so that to receive the events which are necessary for checking the rules assigned to it and notify rule violations to the event broker so that the interested PVCs will be informed about them. After detecting rule violations, monitors “publish” them to event brokers which subsequently forward the violation notifications to the PVCs of the relevant peers subject to existing subscriptions. A detailed description of the monitors is beyond the scope of this paper and may be found in [26].

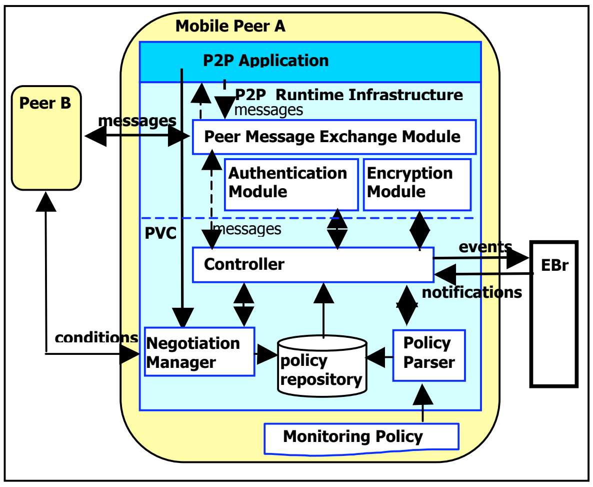

Figure 2. Peer verification controllers

The event broker (EBr) offers the publish/subscribe infrastructure needed for transmitting events from peers to monitors and monitoring results from monitors to peers. The use of the publish/subscribe event reporting infrastructure in MPVF has been due to the need to keep the verification framework separate from the actual P2P service and not to overload the peer communication infrastructure with the event transmissions required for monitoring. It is also adequate for systems where peers may come and go quickly and unpredictably. To preserve confidentiality, EBr manipulates encrypted publications, without having access to their actual contents. This is achieved through the use of secret tokens which act as aliases to the actual information exchanged. These tokens give EBr enough information to manage subscription and publication messages without knowing what they refer to.

3 Specification of monitoring policies

The operation of PVC at runtime is driven by the monitoring policy of its host peer. Policies are specified using an XML version of the language shown in Figure 3 and define:

- *monitoring rules* which specify the properties that should be monitored on the peer that owns the policy or external peers which may interact with it (the applicability of a rule to different types of peers is specified by the *AppliesTo* clause in the policy)
- *assumptions* which are used to derive information about the state of the peers which are being monitored and is necessary for monitoring
- the types of events that the peer is allowed to expose to other peers for their monitoring needs (see *event exposition* element in Figure 3),
- a *timeout* value which determines the maximum time that an event can be blocked whilst a PVC waits for monitoring results about it, and
- a *lifetime* value which determines for how long the policy will be valid.

The logic language that MPVF uses to specify monitoring rules and assumptions within policies is *Event Calculus* (EC) [24]. More specifically, rules and assumptions in monitoring policies are specified as logical formulas of the form $B_1 \wedge \ldots \wedge B_n \Rightarrow H$ where B_i are atomic predicates forming the *body* of the formula and *H* is an atomic predicate forming the *head* of it. The meaning of a rule formula of this form is that when the predicates B_i in its body evaluate to *true*, the predicate H in its head must also evaluate to *true*. The meaning of assumption formulas is that when the predicates B_i in the body of an assumption evaluates to *true*, the predicate H in its head can be derived by deduction.

```
Policy policy_name
 [Rule RuleID String RuleFormula <formula>
   Assumptions
    [AssumptionID String
      AssumptionFormula <formula>]*
   AppliesTo <peer-list-type>*
   [Action <action-type>]*
 ]+
 [EventExposition  <event_exposition_type>]*
 Timeout Duration [drop|forward]
 Lifetime [until Date | permanent]

Key:
[x]*: 0 or more occurences of x
[x]+: 1 or more occurences of x
[x|y]: x or y
```

Figure 3. Monitoring policy specification language

The predicates in the bodies and heads of rule or assumption formulas express events that happen at runtime (e.g., the receipt or dispatch of a peer message), conditions which are initiated or terminated by these events known as *fluents*, or conditions about attributes of the events or the peers that generate them. More specifically, event occurrences are represented by the predicate $Happens(e,t,\Re(t_1,t_2))$. This predicate denotes that an instantaneous event *e* occurs at some time point *t* that is within the time range $\Re(t_1,t_2)$. An event *e* is specified by the term *event(_id, _sender, _receiver, _sig, _source)* where (i) *_id* is the unique identifier of the event, (ii) *_sender* is the identifier of the peer that sends the message, (iii) *_receiver* is the identifier of the peer that receives the message, (iv) *_sig* is the signature of the message that the event refers to, and (v) *_source* is the identifier of the peer from which the message is captured.

```
Policy policy-1

   Rule RuleID Rule_1 RuleFormula
      Happens( e(_eID1, _A, _self, _mes, _self), t1, R(t1,t1)) ∧
      mnt:divide1(t1, PDUR) = _TL ∧ HoldsAt(PeriodRequests(_A, _self, _TL, _MesNum), t1)
      ⇒   _MesNum < MaxReq
      Assumptions
      AssumptionID A1 AssumptionFormula
      Happens( e(_eID2, _A, _self, _mes, _self), t1, R(t1,t1)) ∧ mnt:divide1(t1, PDUR) = _TL ∧
      HoldsAt(PeriodRequests(_A,  _self, _TL, _Req1), t1) ⇒
      Initiates(e(_eID2, _A, _self, _mes, _self), PeriodRequests(_A, _self, _TL, mnt:add(_MesNum, 1)), t1+1)
      AppliesTo   _self
      Actions notify(_eID1, _self)


   Rule RuleID Rule_2 RuleFormula
      Happens( e(_eID1, _self, _A, _mes, _self), t , R(t,t))  ⇒
      (t ≤T_min) and (T_max≤t)
      AppliesTo   _self, TradingPeer
      Actions drop(_eID1, _self)

   EventExposition
   Timeout 1000 drop
   Lifetime permanent
```

Figure 4. Policy example

The initiation or termination of a fluent *f* due to the occurrence of an event *e* at some time *t* is expressed by the predicates *Initiates(e,f,t)* and *Terminates(e,f,t)*, respectively. Two additional predicates, namely *Initially(f)* and *HoldsAt(f,t)*, are used to denote that a fluent *f* holds at the start of the operation of a system and that *f* holds at time *t*, respectively. In the MPVF policy language, fluents are specified as $<relation>(v_1,\ldots,v_m)$ where each v_i is a typed variable or constant value and *<relation>* is the name of the relation.

An example of a monitoring policy specified in the MPVF policy language is shown in Figure 4. *Rule_1* in this policy checks if the total number of requests which are sent to a peer *_self* by the same external peer *_A* over a time period of length PDUR does not exceed a preset threshold value *MaxReq*[1]. The fluent *PeriodRequests(_A, _self, _TL, _MesNum)* in this rule keeps the number of messages (*_MesNum*) that a peer *_A* has sent to *_self* in the time period *_TL*. The value of this fluent is updated when new messages are sent by *_A* to *_self* using the assumption *A1*. More specifically, according to *A1* when a new message is sent to *_self* by *_A* within a particular period *_TL*, the value of the fluent *PeriodRequests(_A, _self, _TL, _MesNum)* is increased by 1 (by virtue of initiating the fluent to the increased value). After the fluent *PeriodRequests(_A, _self, _TL, _MesNum)* is initiated to a new *_MesNum* value, the monitor will be able to deduce the predicate *HoldsAt(PeriodRequests(_A, _self, _TL, _MesNum), t1)* at runtime using an axiom of EC which states that a fluent will hold at any time point after its latest initialisation (unless it has been terminated in between).

MPVF policies also specify control actions that should be executed by the PVC when specific rules are violated. MPVF supports two types of control actions, namely *drop* and *violation notification* actions. Drop actions prevent the dispatch or receipt of the peer message that has caused the violation of a rule and are specified as *drop(eID, $peerID_1$, …, $peerID_n$)*, where *eID* is the identifier of the event that is involved in the violation of the rule, and *$peerID_1$, …, $peerID_n$* are the identifiers of the peers that should be notified of the dropped message. Violation notification actions have as a result the dispatch of a message that notifies a given set of peers of the violation of a rule. These actions are specified as *ViolationNotification(evID, $peerID_1$, …, $peerID_n$)* where *evID* and *$peerID_1$, …, $peerID_n$* have the same meaning a in the case of the drop action above. In the policy of Figure 4, for example, a notification action is specified for *Rule_1* and, thus, when this rule is violated, PVC should send a message to notify the violation to the peer that owns the policy. In the case of *Rule_2*, the action to be applied is to drop the relevant message.

4 Activation of monitoring policies and control

The monitoring policy specified in a given peer is activated as soon as the operation of the peer starts. At this point, the PVC of the peer checks for the existence of a policy on the peer and if such a policy exists, it identifies a monitor and an event broker, sends the rules that the policy specifies for the peer to the monitor, and creates subscriptions in the event broker than will enable the transmission of events that it will generate to the monitor and the monitoring results that the monitor will generate in the opposite direction. At this point the PVC also creates the list of events that it should capture from its peer during its operation and send to the monitor, as well as the actions that should be applied to the events that violate a rule.

As discussed in Section 3, MPVF allows peers to specify in their policies rules that should be monitored not only on themselves but also on other peers. This is possible by declaring the roles of other peers that a rule should be monitored on using the *AppliesTo* clause in a rule specification. This capability is useful when peers want to ensure themselves that their coordinators do not violate rules which are important to them. It is also useful in cases where certain peers in a P2P network are positioned to undertake special responsibilities which require them to oversee certain interactions between all other peers in the network. Peers with special responsibilities may exist due to the use of some form of hierarchy in the peer network (e.g. some peers may act as authorisation entities controlling the admission of other peers to the network) or due to organisational structures that are reflected in the peer network (special peers, for example, may represent team leaders or managers in the real world, who are expected to maintain a higher authority inside the P2P service).

In our example of the enterprise IM application, the manager of a team of normal trader peers may, for instance, wish to monitor rules requiring that none of these normal peers is allowed to send any message during peek trading hours or send trading screens to peers outside its own group of traders. Monitoring such rules in MPVF is possible by including the rules in the policy of the peer which is the group leader and declaring that these rules apply to the roles of the other peers in the

[1] The current period is identified by the execution of the built-in function of the monitor *mnt:divide1(t1,PDUR)* which returns the ceiling of the result of dividing the timestamp t1 of the event by the duration of the periods over which the number of requests is measured (PDUR).

relevant group. *Rule_2* in Figure 4 is an example of such a rule. This rule can be used to monitor message exchanges within peek trading hours as it would be violated if a trading peer sends any message to another peer within the time period from T_{min} to T_{max}. It should be noted, however, that the specification of the external peers which the rules in the policy of a given peer should be applied to is not by itself sufficient for enabling external monitoring. External peer monitoring also requires that the external peers agree to capture and send all the events which are relevant for the given policy to the relevant monitor. This agreement can only result from a negotiation between the involved peers. The peer negotiation process is realised by MPVF as described below.

A peer P_1 starts the negotiation process with another peer P_2 if it needs events from P_2 in order to monitor rules in its own policy that apply to the role of P_2. The negotiation process between the two peers will be triggered the first time that P_1 becomes aware of the existence of P_2 in the P2P network. This happens when P_1 receives a message from P_2 for the first time. When this happens, the PVC of P_1 will identify the rules in its policy that apply to the role of P_2, and then the events that it will need from P_2 in order to check these rules. It will also retrieve the actions that should be executed if the rules are violated. Using this information, the PVC of P_1 will construct a *condition list* of the form

$$[(\text{ev-type}_{i,}\ ((\text{rule}_1, (\text{action}_{11}, \ldots, \text{action}_{1L})), \ldots, (\text{rule}_n, (\text{action}_{n1}, \ldots, \text{action}_{nM}))))] \quad (i=1,\ldots,k) \qquad (1)$$

and send it to P_2 for approval. An element *i* in this list indicates the type of events of P_2 that will be required (*ev-type$_i$)*, the rules against which events of this type will be checked (*rule$_1$*,….,*rule$_n$*), and the actions that should be executed if one of these rules is violated (e.g., *action$_{11}$,…,action$_{1L}$* for *rule$_1$*). After receiving the condition list, the PVC of P_2 will check it against the event exposition specification of its own policy and, if this specification permits the acceptance of the conditions of P_1, P_2 will confirm the acceptance of the conditions and update its internal active policy so as to send the required events to P_1.

Continuing with our previous example, assuming that the role of P_2 is *TradingPeer*, P_1 will need to monitor whether the operation of P_2 is compliant with *Rule_2*. From this rule, it will construct the following condition list and send it to P_2 for negotiation:

[(e(_eID1, P_2, _A, _mes, P_2), ((Rule_2, (drop(_eID1, P_2, P_1))))]

This list will be generated from *Rule_2* after extracting the events referenced by the rule and replacing the variable *_self* in the rule with the identifier of P_2 as the latter peer will become the subject of monitoring in this case. Also in the drop action of the rule the id of P_1 will be added by default since P_1 needs also to be notified by violations of the rule in P_2. Following the receipt of the above condition list, if the event exposition specification in P_2's policy is

EventExposition (e(_id,_self,_any,MSGT1,_self), [TraderGroupLeader], [drop(_mes, ANY)])

P_2 will not accept the condition list of P_1 and negotiation will fail. This is because, according to its exposition specification, P_2 will be able to send only messages of type MSGT1 to peers of the role *TradingGroupLeader* (and, therefore, P_1) but not any event as requested by P_1. If, however, P_2's exposition list had allowed the exposition of all the messages send by it to an external peer, the negotiation would have been completed successfully.

After the conditions are accepted in the negotiation process, P_1 will establish two confidential communication channels to allow the PVC of P_2 to send the events required for monitoring to the monitor of P_1 and the monitor to notify the results of the monitoring process back to P_1 and P_2. In MPVF these communication channels are identified by tokens generated by P_1. MPVF assumes that event brokers are not trusted entities and therefore they should be able to manage the subscriptions and publications of events and monitoring results without having access to their contents. To achieve this, the events and monitoring results are encrypted and the necessary keys for the decryption of this information are generated outside the event broker and are not made available to it. The event broker gets only tokens that identify the notification channels and enable it to distribute the encrypted messages to the appropriate subscribers. Tokens essentially provide aliases to the actual information exchanged, giving the event broker sufficient information for managing subscriptions and routing publications, without knowing what a token refers to and being able to deduce the actual type of the transmitted messages or other information from it. The protocol for creating the tokens and decryption keys and establishing the event and notification reporting channels is based on SSL [11]. A detailed discussion of this protocol is beyond the scope of this paper and may be found in [18].

After the activation of a policy in a peer, its PVC controller catches the messages sent to and from it and finds the set of rules that need to be checked for each of these messages and have a drop action defined for it. If there is no rule with a drop action for the message, the controller transmits an event representing the message to the event broker without waiting for any monitoring results. If, however, there are rules with drop actions, then the controller must ensure that all these rules are satisfied before allowing the message to be transmitted to its destination. Thus, the controller blocks the message and waits for notifications of monitoring results from the monitor (via the event broker). Whilst waiting for these notifications, if a timeout occurs, the controller forwards the message to its destination peer. In the case, however, whwre a violation of a rule with a drop action for the message is notified before a timeout, the controller drops the message and stops waiting for any further notifications of monitoring results for the message as these can be handled by the notification handling process of PVC in an asynchronous mode. The PVC controller will release a message that is checked against rules with drop actions only when it receives notifications from the monitor confirming that none of the relevant rules has been violated so far and cannot be violated by the message in the future.

5 Related work

The work that we have presented in this paper is related to research on security and monitoring of P2P systems and approaches developed to support runtime verification in general.

Relevant strands of work in the former area are related to: (a) P2P system security and trust, including frameworks for computing peer reputation ratings [8,15,25], admission control schemes [10,22], techniques for P2P data exchange encryption [28], and decentralised key management [29], and (b) approaches supporting the monitoring of resources in P2P networks [12]. Some of these frameworks and techniques deploy specialised forms of monitoring focusing mainly on the existence, extent of use and sharing of resources in P2P networks [3,12,15] or detecting application-specific traffic in P2P networks [22] rather than checking the compliance of runtime P2P system operations with specific properties as MPVF does. We should also note that MPVF-based monitoring could be used to generate peer reputation ratings and enforce admission and access control policies in P2P systems.

In the area of runtime verification, there are approaches that support monitoring on different implementation platforms including, for example, Java programs [4, 5, 7, 16, 17] or BPEL workflows [13, 19, 27]. None of these approaches, however, focuses explicitly on mobile P2P systems or provides a framework that can support effectively the verification of such systems by including mechanisms for: (a) generating events from such systems without having to change their code, (b) negotiating monitoring conditions between peers in order to activate monitoring when a P2P system evolves with the admission and departure of peers, and (c) applying control actions in response to certain types of violations. Thus, the framework presented in this paper is novel in addressing exactly these aspects.

6 Conclusions and future work

In this paper, we presented a framework (called MPVF) that we have developed to enable the monitoring of policies of application level security and dependability properties for mobile P2P systems. In addition to monitoring, this framework supports the automatic negotiation between peers at runtime in order to enable the activation of monitoring, the emission of events required for monitoring from peers to the monitors, and the dynamic execution of actions following the detection of property violations.

MPVF has been implemented using the SIENA event notification service [6] and has two implementations: one that is based on JSE v1.5 and a version for mobile phones based on JME-CDC 1.0 which has been tested on Sony Ericsson's P990i. MPVF can be used by P2P applications built upon the PEPERS peer communication framework. To deploy MPVF, developers need to write policies that drive the monitoring activity during the operation of a P2P system and provide information about EBr and the monitor(s) that may be used at runtime as part of a configuration file. However, there is no need for developers to add any extra code to their application unless they want to notify end-users of the monitoring results or take some application specific action in response to them.

Currently, we are investigating the possibility of extending MPVF with a monitor discovery service. In this service, monitors will be treated as a special type of peers that could be discovered dynamically using appropriate P2P protocols.

ACKNOWLEDGEMENTS

The work reported in this paper has been funded by the European Commission under the Information Society Technologies Programme as part of the project PEPERS (contract IST-026901).

REFERENCES

1. Androutsellis-Theotokis S., and Spinellis D. (2004), A survey of peer-to-peer content distribution technologies. ACM Computing Surveys, 36(4):335–371
2. Barringer H., Goldberg A., Havelund K., and Sen K. (2004), Rule-Based Runtime Verification, In Proc. of 5th Int. Conf. on Verification, Model Checking, and Abstract Interpretation
3. Binzenhofer A., Kunzmann G. and Henjes, R. (2006), A scalable algorithm to monitor chord-based p2p systems at runtime, In Proc. of 20th Int. Parallel and Distributed Processing Symposium.
4. Brörkens M. and Möller M. (2002a), Dynamic event generation for runtime checking using the JDI, In Proc. of the Federated Logic Conf. Satellite Workshops, Electronic Notes in Theoretical Computer Science, 70 (4)
5. Brörkens M. and Möller M. (2002b), Jassda trace assertions, runtime checking the dynamic of Java programs, In Proc of Int. Conf. on Testing of Communicating Systems, Berlin, Germany, 39-48
6. Carzaniga A., Rosenblum D. S., and Wolf A. L. (2000), Achieving scalability and expressiveness in an internet -scale event notification service, In Proc. of 19th ACM Symposium on Principles of Distributed Computing
7. Chen F. and Roşu G. (2007), MOP: an efficient and generic runtime verification framework. In Proc. of the 22nd ACM SIGPLAN Conf. on Object Oriented Programming Systems and Applications, 569-588
8. Damiani E., di Vimercati D., Paraboschi S., Samarati P., and Violante F., (2002), A reputation-based approach for choosing reliable resources in peer-to-peer networks, In Proc. of 9th ACM Conf. on Computer and Communications Security, 207 – 216
9. D'Amorim M., Havelund K. (2005), Event-based runtime verification of Java programs, In Proc. of 3rd Int. Works. on Dynamic Analysis (WODA'05)

10. Fenkam P., Dustdar S., Kirda E., Reif G. and Gall H., (2002), Towards an access control system for mobile peer-to-peer collaborative environments, WET ICE 2002 – Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises

11. Freier A. O., Karlton P., and Kocher P. C. (1996), The SSL Protocol Version 3.0, Internet draft, available at: http://wp.netscape.com/eng/ssl3/draft302.txt

12. Gedik B. and Ling L. (2003), PeerCQ: A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System, In Proc. of 23rd Int. Conf. on Distributed Computing Systems, 490- 499

13. Ghezzi C., and Guinea S. (2007), Runtime Monitoring in Service Oriented Architectures, In Test and Analysis of Web Services, (eds) Baresi L. & di Nitto E., Springer, 237-264.

14. Groce V., Maggio M., Ficano I., Siveroni I., and Androutsopoulos K. (2007), Framework and Security Verification Tools, Deliverable D6, PEPERS, FP6-26901

15. Gupta M., Judge P., and Ammar M. (2003), A reputation system for peer-to-peer networks. In Proc. of the 13th Int. Works. on Network and Operating Systems Support For Digital Audio and Video

16. Havelund K., and Roşu G. (2004), An Overview of the Runtime Verification Tool Java PathExplorer, Formal Methods Syst. Des. 24: 189-215

17. Kim M., Kannan S., Lee I., Sokolsky O. and Viswanathan M. (2001), Java-mac: a run-time assurance tool for Java programs, In Electronic Notes in Theoretical Computer Science, 55. Elsevier Science Publishers

18. Koulouris T, Tsigkritis T. and Spanoudakis G. (2006), Dynamic Verification Support Framework, Deliverable D4, PEPERS Project, IST-2004-026901

19. PEPERS, www.pepers.org

20. Pistore M. and Traverso P. (2007), Assumption Based Composition and Monitoring of Web Services, In Test and Analysis of Web Services, (eds) Baresi L. & di Nitto E., Springer Verlang, 307-335.

21. Reuters Messaging, http://about.reuters.com/productinfo/messaging/?user=1&

22. Saxena N., Tsudik G., and Yi J.H. (2003), Admission control in Peer-to-Peer: design and performance evaluation, In Proc. of 1st ACM Workshop on Security of Ad Hoc and Sensor Networks, 104-113

23. Sen S., Spatscheck O., and Wang D. (2004), Accurate, scalable in-network identification of p2p traffic using application signatures. In Proc. of the 13th Int. Conf. on World Wide Web

24. Shanahan M.P. (1999), The Event Calculus Explained, Artificial Intelligence Today, LNAI 1600:409-430

25. Song S., Hwang K., Zhou R. and Kwok Y. (2005), Trusted P2P Transactions with Fuzzy Reputation Aggregation. IEEE Internet Computing, 9(6):24-34.

26. Spanoudakis G. and Mahbub K. (2006), Non Intrusive Monitoring of Service Based Systems, International Journal of Cooperative Information Systems, 15(3):325-358

27. Mahbub K., and Spanoudakis G. (2007), Monitoring WS-Agreements: An Event Calculus Based Approach, In Test and Analysis of Web Services, (eds) Baresi L. & di Nitto E., Springer Verlang, 265-306.

28. Xiaolin Catania, B. and Kian-Lee T. (2003), Securing your data in agent-based P2P systems, In Proc. of 8th Int. Conf. on Database Systems for Advanced Applications, 55- 62

29. Law Y.W., Corin R., Etalle S. and Hartel P. (2003), A Formally Verified Decentralized Key Management Architecture for Wireless Sensor Networks, Personal Wireless Communications, LNCS 2775: 27-39

30. Mobile P2P messaging, http://www.ibm.com/developerworks/wireless/library/wi-p2pmsg/#9 (last seen on 27/2/08)

31. Navizon, P2P Wireless Positioning, http://www.navizon.com/ (last seen on 27/2/08)

32. PeerBox, http://www.peerboxm.com/Default.aspx? (last seen on 27/2/08)