CMU-PT/RNQ/0015/2009

Trustworthy and Resilient Operations in a Network Environment

Deliverable D10

First Specification of the Architecture

Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL
AN INTERNATIONAL PARTNERSHIP

Executed by: **LaSIGE**

Direction/Department: **DI/FCUL/UL**

Date: **04-11-2011**

Version Number	Owner	Change Control	Date
0.1	FCUL	Document creation with the first draft specification of the architecture	14-10-2011
0.2	FCUL	First complete review with many structural and content fixes and improvements	21-10-2011
0.3	FCUL	Formatting, structural and additional information improvements	21-10-2011
0.4	FCUL	Second review with further improvements, mostly on sections one, two and four	28-10-2011
0.5	FCUL	Improvements on the specification of the architecture and some minor reviews	02-11-2011
1.0	FCUL	First peer reviewed and improved version of the first specification of the architecture	04-11-2011

Additional information:

Author/s:	Smruti Padhy, Diego Kreutz, António Casimiro, Marcelo Pasin
Contributor/s:	Bruno Sousa, Filipe Araujo, Marília Curado
Beneficiaries contributing on the deliverable:	FCUL, FCTUC
WP contributing to the deliverable:	WP3
Nature of the deliverable:	Report
Total number of pages:	50

— Document generated on 4 November 2011 at 21:49 —

Contents

List of Figures

List of Tables

Executive Summary

This project aims to develop a fault and intrusion tolerant framework for monitoring cloud infrastructures in a trustworthy way. It targets typical data center scenarios as those of Portugal Telecom. The framework will improve resilience and guarantee the needed trustworthiness for monitoring cloud infrastructures. The PT data center specific needs will be our real application case used to validate the proposed fault and intrusion tolerant monitoring architecture and, at the same time, improve the company monitoring services.

This document introduces the First Specification of the Architecture for the Fault and Intrusion Tolerant system considered in TRONE project. The presented architecture results from a study of related works, protocols and technologies that are useful to solve the stated problem of improving cloud infrastructure monitoring systems resiliency and trustworthiness.

Current security analyzers and monitoring systems for cloud infrastructures, such as HP OpenView and ArcSight, are based on local, centralized or hierarchical model approaches. Additionally, they do not look deep into resilience and delivering trustworthy data of its own services under crash or Byzantine failures caused by attackers or by any other kind of sources. In this deliverable, we design and present architecture of a fault and intrusion tolerant monitoring system for cloud computing infrastructures. We assume a Byzantine failure model and propose state machine replication for providing the trustworthy and resilient monitoring service. We propose a publish-subscribe system for event dissemination where distributed probes publish monitoring event messages into the system and monitoring consoles subscribe the service for receiving specific information. The fault and intrusion tolerant monitoring system will deliver event messages from probes to consoles in a reliable and trustworthy way.

The proposed architecture is intended to fit the considered scenario, that is, it will improve trustworthiness and reliable monitoring of infrastructures of PT's data center. However, the architecture is generic enough, so that it could also be used in other network operations scenarios and also to solve other problems such as the heterogeneity of security analyzers and monitoring tools available for networks and services. Those different tools may become clients of the system, receiving data from a huge variety of probes and agents distributed across networked resources and systems.

This deliverable is structured as follows. After a brief introduction and motivation in Chapter 1, we provide a short description of related works, including cloud monitoring and security analyzer tools, Byzantine fault tolerant protocols, publish-subscribe systems and resilience through multihoming in Chapter 2. Then, we present the system model in Chapter 3 describing network, synchrony and fault models. Chapter 4 describes the fault and intrusion

tolerant monitoring system architecture. Details about the event service and its interfaces are given. Further, we give examples of TRONE-aware applications such as multihoming-based applications, security log analyzers and monitoring systems in Chapter 5. Finally, we present some conclusions and future steps to be done within the project.

Chapter 1

Introduction

Cloud computing is becoming popular mainly because of the vast available computing resources on clouds at affordable price and hassle free installation. Several data centers are finding business in providing cloud computing services. A typical data center deploys several machines to provide Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). In addition to these, it has monitoring and management services for its internal network such as LDAP [66, 52], DNS [9], DHCP [22], SMTP [38], SNMP [11], Puppet [55] and CFEngine [13]. These are mostly used for maintaining different features and resources needed to provide effective cloud computing solutions to the end user. These services are responsible for keeping the correct operation, availability, performance and security of the cloud infrastructure.

In data centers and companies it is common to have separated VLANs for each kind of critical applications. For instance, virtual machines (VMs) running control and management services are usually connected on a separate network, for security reasons. However, virtual networks may represent a security threat once they are bypassed, which might happen due to misconfigurations or hopping attacks [49].

Another problem that may arise with different virtual networks is the potentially relaxed system security cautions taken by network operators. As they rely on the virtual network isolation theoretical security, they are less likely to care about improving the security of the systems running inside a particular VLAN. This could lead to highly risky attack prone scenarios. Furthermore, there are also system software bugs and breaches that can be a turn key for attackers. Inside these kinds of environment, critical virtual machines running monitoring services are attack prone. If one of these machines is compromised by the action of an attack, all monitoring and control systems and information are not reliable anymore,

potentially compromising the overall computing infrastructure operation and management. In the worst case, a hacked monitoring system could lead to computing components functional faults, critical infrastructure data theft, service level agreement problems. This could also involve harder and time consuming effort for operation and management diagnosis, as the console information may have been faked by hackers.

A typical IaaS environment is composed of many computing components such as processing servers, high availability gateways and servers, high speed switches, routers, storage systems, cloud controllers, firewall and monitoring and control appliances and systems. The servers, for instance, are commonly used to maintain many virtual machines, by using virtual machine monitor software. This kind of infrastructure resides inside data centers and usually makes use of virtual networks to isolate different data traffics. Maintaining and ensuring correct operation of these computing components and the overall infrastructure in data centers are really complex and tricky tasks to be accomplished.

IaaS providers use resources such as probes, consoles and monitoring systems to take care and control heterogeneous, complex and critical environment, such as cloud infrastructures. A monitoring system is a key component in data centers. It allows network operators and managers to discover, diagnose and foresee problems in computing components, being able to take further actions for fixing or avoiding problems. A typical monitoring infrastructure is composed of probes collecting data from the different computing components. The probes can reside inside or outside the computing components. Their main task is to generate input data for the monitoring systems. On the other side, we have the consoles representing the interface between the monitoring systems and the end users. All events generated by probes, and collected by monitoring systems, are sent to the consoles and, subsequently, shown to the network operators and managers by means of statistics, graphics, dashboard, alerts, messages, along with other different data representation forms.

In a monitoring infrastructure, probes and consoles can be easily replicated, increasing fault tolerance. One computing component can have two or more probes observing its behavior. The data displayed to network administrators can be easily presented by several replicated consoles. However, the monitoring system, such as ArcSight [33] security threat analyzer engine, is still a single point of failure. Replicating monitoring systems to achieve fault tolerance is not an easy task. Unlike probes and consoles, the monitoring systems are big and have more complex components, thus, less likely to be replicated. Most of existing event message forwarders and security threat analyzers are designed to work in a centralized way. This means that those tools are single points of failure and can be more easily attacked and compromised. Thus, there is space for further investigation and development of monitoring

frameworks intended to propagate event messages from probes to consoles in a trustworthy way.

Considering these real environments, scenarios and problems, TRONE project aims to design and develop a fault and intrusion resilient monitoring system. To achieve this goal, we propose a new fault and intrusion tolerant monitoring system architecture for cloud infrastructure. IaaS are specific and critical computing infrastructures spreading world wide to provide platform and software as a service for end users. This is the case at Portugal Telecom (PT), where they are deploying infrastructure on demand products such as SmartCloudPT [54]. Figure 1.1 illustrates such a cloud infrastructure environment. Computing systems are isolated, by different VLANs, from monitoring and control systems, as is the case at PT.

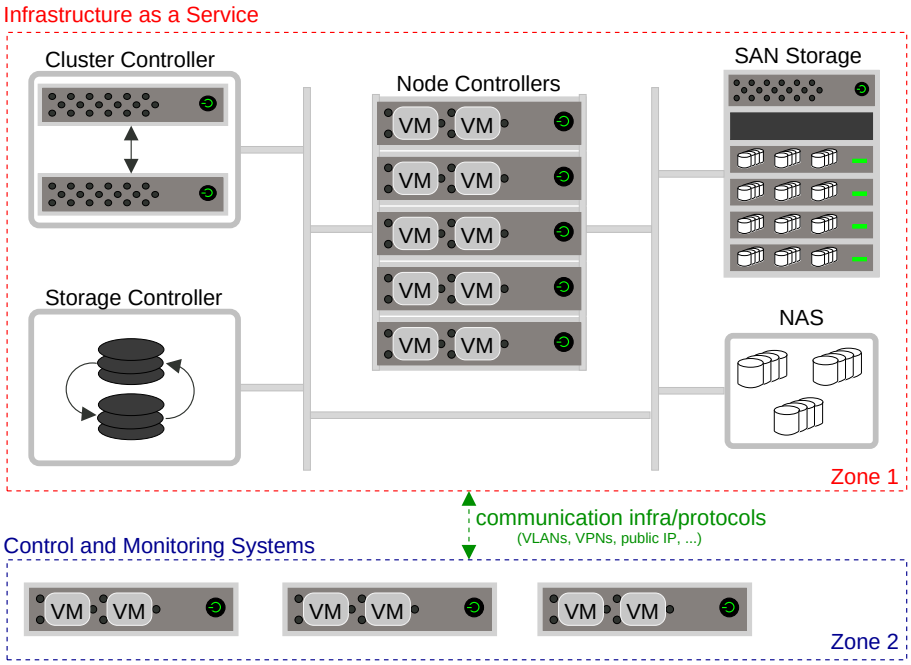

Figure 1.1: A cloud infrastructure scenario with network isolated monitoring systems

We plan to evaluate the fault and intrusion tolerant architecture using probes and consoles for monitoring IaaS. We intend to use currently available probes from monitoring and log analyzer tools such as OSSIM, Snort and ArcSight, and also develop specific probes for cloud infrastructure. As consoles, we are going to try to use PT's available consoles as well as develop simples software version consoles. Furthermore, we intend to use as subscribers of the monitoring system tools such as ArcSight, CorreLog and Pulso.

1.1 Motivation

A typical cloud infrastructure monitoring system comprises probes or agents connected to a console through a centralized event message handling system. One existing solution is ArcSight Enterprise Security Manager (ESM). It works in a centralized way collecting mainly security related data, basically logs, from various devices and applications, generating alerts on the occurrence of critical events. It has a correlation engine to analyze events based on their context. Portugal Telecom, the major telecommunication operator in Portugal, is using ArcSight for monitoring their infrastructure's security. PT envisions the use of cloud in public critical infrastructures such as grid, health care and other strategic applications. Concerns about quality of service (QoS) and quality of protection (QoP) when using IaaS for these environments become very critical because companies and people rely their services on cloud infrastructures expecting 100% of uptime. Thus, they need to be monitored in a resilient and trustworthy way for security and availability.

At PT's environment, devices and systems under monitoring are connected to a console through a central ArcSight appliance. This centralized monitoring system becomes a single point of failure or attack. If the system is compromised it will affect the overall computing infrastructure monitoring and control.

Figure 1.2 shows this abstraction of the monitoring system being used in PT. A single system can be monitored from different perspectives. Each console shows data of different systems or different perspectives of a same system. Such a console can also be a single point of attack or failure. To avoid such single point of failure or attack, we can have several consoles for each perspective and system. To maintain consistency of monitoring results between consoles and also to ensure trustworthiness of the events reported by probes, we propose a fault and intrusion tolerant (FIT) monitoring system in the middle, between the system being monitored and console.

Figure 1.3 gives an overview of the proposed monitoring system. As can be seen, it should propagate the probe's event messages to consoles in a trustworthy way. However, FIT monitoring system itself, such as ArcSight, can also be a single point of failure or attack. So, as shown in figure 1.4, we propose to design a FIT monitor with replication in mind to ensure the trustworthiness and availability of the monitoring system itself.

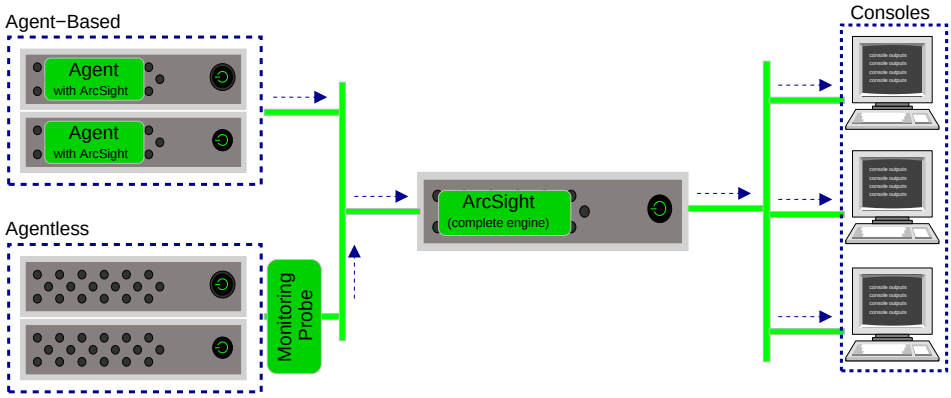

Figure 1.2: Abstraction of the existing monitoring system at PT

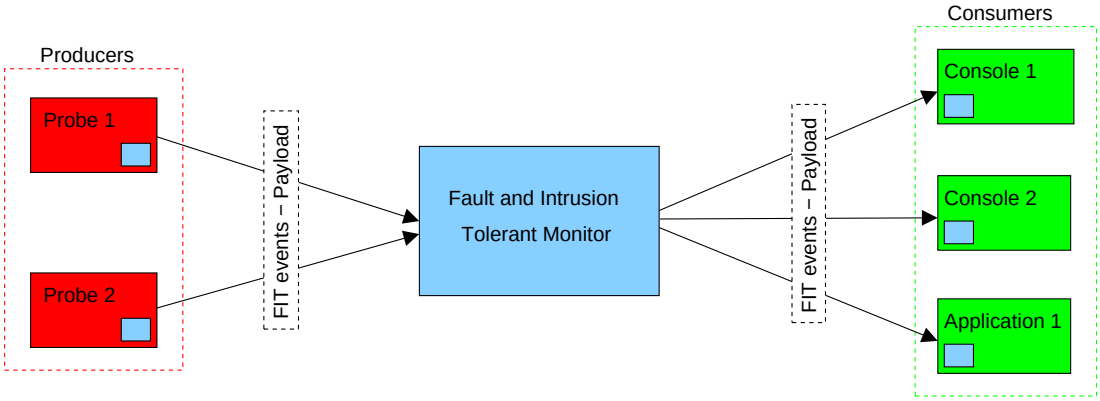

Figure 1.3: High level abstraction of the proposed monitoring system

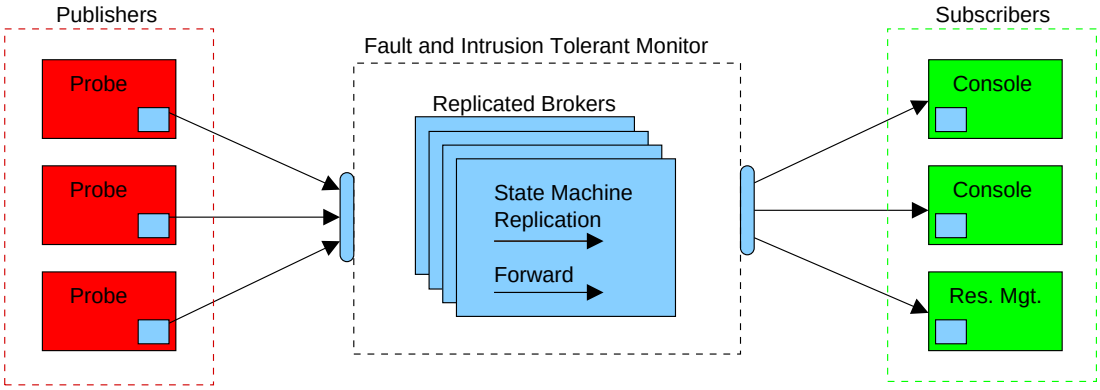

Figure 1.4: FIT monitoring system improved reliability with replication

1.2 Problem statement

The aim of this deliverable is to propose an architecture and protocols for a cloud infrastructure monitoring system designed with fault and intrusion tolerance capabilities. The monitoring system should be resilient to any kind of faults, allowing messages to go from

probes to consoles in a trustworthy way.

To address this problem, we propose a topic-based event-based Publish-Subscribe (P-S) system for event message dissemination. Probes become publishers while consoles act as subscribers. The FIT monitor works as a reliable framework for delivering data from publishers to subscribers. To ensure Byzantine fault tolerance (BFT) in FIT monitoring we propose the use of State Machine Replication (SMR). This makes the monitoring system resilient to any kind of faults, both crash or Byzantine. On the other hand, it requires the replicated system to be deterministic.

Chapter 2

Related Work

In this chapter we briefly describe the state of the art for tools and algorithms related with the TRONE project goal. In sections 2.1 and 2.2, we present some cloud infrastructure monitoring and security information event managers. These tools are critical to maintain current and future IaaS. We give examples of publish-subscribe systems (P-S) in section 2.3. These systems facilitate the communication between between a set of producers, a set of consumers and event broker. Producers are naturally content publishers while consumers become content subscribers. Later, we describe Byzantine fault tolerant protocols in section 2.4. They provide the means for systems that need to tolerate any arbitrary fault. Further, a brief introduction to multihoming is presented in section 2.5. It is the basis for one of the proposed TRONE-aware applications. Finally, we conclude the chapter with some short final remarks.

2.1 Cloud infrastructure monitoring tools

In this section we present some of the most widely known tools available for cloud computing infrastructure monitoring. They are designed with different concepts and features in mind. However, all of them have as final goal to meet business needs by providing online information about the IaaS running components.

Cloudkick [16, 19] monitor is a service available to customers for managing their servers. It has a good set of built-in checks and also allows customers to execute simple monitoring scripts, which are plugins written in any language, used by local agents to perform specific tasks. This allows cloud owners to exactly do what they want because almost anything can be included inside the Cloudkick monitoring solution. But its flexibility comes with a price.

End users should be aware of the risks and pitfalls when creating specific designed monitoring scripts. One of the interesting things about Cloudkick is its design as a service. It even has a free online try. Some of the monitoring checks are available for free use. Other ones have to be paid. The customer actually connects the Cloudkick service with its cloud computing infrastructure freeing the hassle of installing and configuring a new tool into his domain. Only the monitoring agents have to be installed into the corresponding cloud components that are being monitored.

Zenoss Unified Monitoring [70, 69] is designed to save customers time in the task of monitoring the entire physical and virtual IT environments from a single point, providing in a dashboard accurate information about the state of every component being monitored. The tool provides a big picture of the health of customer infrastructures, including hardware, virtual machines and their corresponding applications. It unifies event and performance data in one console, doing cross-discipline analytics to identify the big hits for most appropriate infrastructure investments. It tries to add integrated support for technologies of a dynamic data center [70]. This includes solutions as Cisco UC, Cisco UCS, VMware vSphere, VMware vCloud, and NetApp. Zenoss customization for end users relies mostly over personalized dashboards and views of the monitoring data, focusing on easing infrastructure management. The solution also provides interfaces for integrating it into external portals or dashboards.

Amazon CloudWatch [4] is a tool specifically designed to monitor Amazon cloud computing components like Amazon EC2 instances, EBS volumes and RDS DB instances. It also allows customers to collect specific data from their running applications and services. The basic idea is to provide system administrator with resources to collect and track metrics, gain insights and react immediately to avoid business hassle. The Amazon CloudWath is a cloud service available for Amazon EC2 clients. It provides customers basic monitoring for free and paid detailed monitoring. Users are allowed to use the Amazon CloudWatch API into their applications, being a way of providing customized monitoring metrics. Data is provided to customers by means of graphs, statistics and alarms through AWS Management Console.

VMware vFabric Hyperic [64] is the application monitoring component of the VMware vFabric Cloud Application Platform [65]. It is intended to provide performance and availability of Web applications in physical, virtual and cloud environments through its more than 50.000 metrics across 75 application technologies. Besides the pre-defined capabilities, it can be extended to monitor any component in the customers' application stack. With the Hyperic dashboards and reports, users may monitor and demonstrate compliance with SLAs, operating level agreements (OLAs), and underpinning contracts (UCs) to their respective owners

and IT management. This is another interesting management feature needed and/or desired by many client companies. VMware vFabric Hyperic works with operating systems such as Linux, Windows, Solaris, HP/UX, AIX, BSD family and Mac OS X. It works with virtualization platforms like VMware vSphere, vCenter, vCloud Director, and Xen VM. Hyperic also works with Java platforms and application servers among other technologies. And, finally, it provides integration with other monitoring tools such as Nagios, OpenNMS, SNMP to OpenView, Tivoli and Patrol.

LogicMonitor [41] is an IaaS monitoring tool that works with SNMP, JMX for native Java applications, JDBC for databases, WMI, and more. It provides native support for Citrix XenServer, VMware vSphere and ESX, Amazon EC2 and Eucalyptus. LogicMonitor is able to discover IaaS components to be monitored. After being discovered they will be under the monitoring system, providing for users a complete visibility of they infrastructure in an automatic way. Through flexible dashboards, it allows managers to correlate performance data, providing means to identify bottlenecks and proceed infrastructure arrangements.

Monitis platform [47] is a service provided by Monitis company. It is provided as a cloud-based service. Monitis consists of a central server, global monitoring network, highly scalable databases, remote probe agents, and graphical user interface applications, working as a featured and ready to use solution for right away monitoring IT infrastructures. The platform is capable of managing different IT assets ranging from simple/complex IP devices to web sites, applications and servers.

The Monitis agents are available for multiple platforms. They use HTTPS polling for connection with the central server, so no changes should be performed on the firewalls. Agents send data to the central server [47].

Nimsoft Monitor [50, 51, 36] is a solution available either on demand (SaaS) or for local installation. It is designed to proactively monitor and manage performance, events, and alarms on both traditional data centers and newer virtualization and cloud.

CloudSec [3] explores the security problem of the IaaS cloud computing model. It is a security monitoring appliance designed to provide active, transparent and real-time security monitoring for hosted virtual machines in the infrastructure as a service model. The tool makes use of machine introspection techniques for dynamically monitoring the physical memory of VMware ESX virtual machines. CloudSec monitors the changing kernel data structures looking for malwares through the effective detection of user or kernel rootkits. In this sense, it works specifically with individual Virtual Machine Monitors (VMMs), monitoring its corresponding virtual machines. One challenge is to transport monitoring data in a secure way

to the monitoring consoles of systems managed. This is an important task that needs to be accomplished to ensure that viewed data is reliable and trustworthy.

2.2 Security information event managers

Security information event managers (SIEMs) are important tools to provide real-time security alerts analysis. They are able to process and correlate security logs and events at a centralized system. Probes and agents may gather security data and information from different systems and devices, sending them to a central server where a SIEM tool is going to proceed the further analysis. This is nowadays an important way for companies to improve their detection, diagnose and response time over security threats.

Some of the existing and most widely SIEM tools are OSSIM (Alien Vault Unified SIEM), SALSA, Feedzai, SCOM, Pulso, ArchSight SIEM and Tivoli SIEM. Each of them is explained in this section.

The main goal of OSSIM is to do event correlation with the intention of detecting any threat and monitor security events. In its architecture, the sensors detect and collect security incidents data. To do so they are distributed within the infrastructure systems. Inside OSSIM [1] architecture there is a SIEM component that provides the means to perform risk assessment, event correlation, risk metrics analysis and vulnerability detection. It uses a database with normalized information, collected mainly from system loggers, to proceed data mining and security intelligence algorithms. Results of proceeded analysis are presented to users through dashboard, alerts, among other means.

SALSA [5] is an analysis tool designed to automate the processing of system log files. It does control and data flow execution tracing in a distributed system. The tool also retrieves the system state on each node. SALSA uses additional semantics to analyze event-based logs to identify for how long the system is performing a specific activity. It tries to identify the key activities inside system logs by the means of start and end times of specific events. Another key application of SALSA is related to failure diagnosis. It analyzes the behavior of failure-free and failure-injected hosts, producing enough data patterns to hypothesize what failures could be diagnosed by using probability distributions of failure durations.

Feedzai Pulse [27] is designed to provide real-time event analysis over huge amount of data produced by systems inside enterprise level companies. With a high level architecture in mind, it has been designed to be used in many applications for online event-driven data

analysis. Because of this, it is able to receive and process security events from different sources. Based on pre-defined rules, it will perform security analysis over the incoming data.

The system center operations manager (SCOM) [43] is designed to aggregate all events generated by different Microsoft components. A system manager defines rules which are used by the tool to analyze incoming events and take actions like activate alarms based on identified events that potentially characterize a critical problem or a threat. SCOM has a specific module, called Audit Collection Services (ACS), intended to identify security threats or even lack of security compliance such as hacking and viral activities, resources misuse and attacks.

Pulso [20] has a client-server architecture. All clients, distributed agents, collect data and send it to a central repository. There a correlation engine processes the data trying to identity systems QoS status, security breaches, system quality and QoM (Quality of Management). The reports provided by Pulso may be used on more high level meetings as well as by technical teams.

ArcSight Enterprise Security Manager [33] is a product of HP company for monitoring enterprise threats and risks. It correlates the events occurring within the enterprise, interpret them and raise security alerts if there is a security breach. It provides ability to collects logs from over 300 devices and event sources such as OS, routers, switches and storage systems. The tool is designed to process a huge amount of events per second. It is one of PT's current choice system for receiving and analyzing devices and system logs of their computing infrastructure. They mainly use the tool intending to find and take actions over potential security threats.

Tivoli SIEM [34] is designed to work with IBM Tilovi line of products. It has as goal to facilitate compliance efforts with centralized dashboards and highly developed reporting capabilities. It is composed by components that enable to investigate and retrieve native system logs. Tivoli SIEM works with real time analytics in mind trying to understand and alert on insider threats.

There are other more specific tools such as QuIDScor [56] and Snort Correlation Engine [48]. These tools are specialized for correlating events from intrusion detection systems such as Snort.

LCE stats for Tenable Log Correlation Engine [60]. This system normalizes and analyzes logs from network devices. It is able to analyze and process data from different sources such as firewalls, intrusion detection and prevention systems, network traffics, system and application logs and user activity. The correlation engine searches in real-time for threats and vulnerabilities.

There are also other tools like CorreLog [17] and SEC [61]. The former is a specialized high performance log analyzer designed to correlate security events of different sources from a variety of platforms. It uses neural network technology within system rules to output the impact of future behaviors, trying to predict future systems behaviors, generating alerts and notifications when ever necessary. And the further tool is a simple event correlator. The main goal is to provide a less complex and cumbersome tool to smaller event correlation tasks.

2.3 Publish-Subscribe systems

Publish-subscribe systems are one natural way to solve the problem stated when we have producers and consumers. The former may publish event messages into a messaging system while the latter may subscribe for specific contents. The messaging middleware is responsible for receiving event messages from publishers and send them to the interested subscribers. This works well for a monitoring system framework where we have probes (publishers) and consoles (subscribers).

There are researches that provide classifications and comparisons of many available variants of publish-subscribe system based on decoupling of communicating entities with respect to time, space and synchronization [25, 40]. They also provide a systematic arrangement based on the expressiveness of the search (filter) methods, such as topic-based, content-based and type-based. Some implementation issues involved in such paradigm are also discussed [25, 40].

There are several works on search methods for efficient filtering and matching algorithms [26, 53, 32], distributed event routing [7] and adaptive routing to traffic demands [44] in publish-subscribe systems. However, only a few works addressed the fault-tolerance in P-S system. One research proposed a publish-subscribe algorithm that tolerates up to δ brokers crash failures [37]. Brokers form a tree-based overlay topology and maintain a topology map which contains a partial view of the topology. Partial views are stored in the form of a tree, with the broker as a root, and with vertices and edges within distance $\delta + 1$ from the broker. The work proposed a topology management, a subscription scheme, a publication forwarding protocol in presence of up to δ failures, a recovery procedure and an optimization of number of network messages.

Another work [58] describes a fault-tolerant and secure service of sealed-bid auctions. The proposed algorithm can tolerate one-third of Byzantine failures of auction servers and any

number of bidder failures. It uses PVS (Prototype Verification System) for formal specification and verification of the system properties. This application maps to a loosely coupled publish-subscribe system.

In the TRONE project the research will focus on Byzantine failure and intrusion tolerant publish-subscribe systems, which is still a subject that needs further investigation and development.

2.4 Byzantine fault tolerant protocols

There is one work [12] that proposed a practical Byzantine fault tolerant protocol, PBFT, which is a form of state machine replication algorithm. The proposed algorithm allows the system to tolerate at most f Byzantine faults within $3f + 1$ replicas and is also suitable for asynchronous systems. It involves three phases in normal case operation (i.e., no faults case): a) pre-prepare; b) prepare; and c) commit. These three phases constitute the agreement protocol of the algorithm which decides the order of execution of requests. To mask failures of the primary, the algorithm uses the view change protocol.

Proactive recovery mechanism for BFT have been proposed [12] to enable system recovery from any number of failures over its lifetime provided that the number of faulty replicas is limited to one-third of total replicas within a window of vulnerability. This work was followed by several others with the aim of optimizing BFT's performance overheads [39, 68, 67].

One recent research proposed the Zyzzyva protocol [39] for reducing the cost and design of BFT state machine replication. The protocol allows replicas to speculatively decide on which order to execute on requests without running the agreement protocol.

There are also works focusing on improving on the minimum number of replicas required to tolerate Byzantine failures. One of them [68] showed that $2f + 1$ replicas may be sufficient to execute requests and tolerate f faults. This is achieved by separating agreement protocols and execution, resulting in $3f + 1$ replicas to order the requests and only $2f + 1$ for execution.

Another work [67] showed that the number of replicas required for execution can be further reduced to $f+1$ with the help of virtualization technology. Another approach for reducing the number replicas have been developed [57]. It proposes a state machine replication algorithm for Byzantine agreement in wide-area networks. A trusted component is used on the servers to reduce the number of replicas and communication steps required for agreement. In doing so, it requires only $2f + 1$ replicas.

Further, a brief BFT protocols state of art can also be found [8]. The author presents solved, partially solved and open problems stating some challenges that still need to be addressed.

The BFT protocols will be an important part of the TRONE project for supporting the development of a reliable and trustworthy monitoring architecture. One way to use Byzantine fault tolerant protocols is through SMaRt [2]. It is a high performance Byzantine-fault-tolerant state machine replication library developed in Java. Its main strengths are simplicity and robustness. Thus, it can be used as a building block in the development and deployment of a fault and intrusion tolerant monitoring system, as proposed in this deliverable.

2.5 Resilience through multihoming

Stream Control Transport Protocol (SCTP) [10] is a connection-oriented protocol designed to assure reliable transport and to support multihoming natively, through different mechanisms. First, via address management at the association setup, during which a node informs its peers about its IP addresses (or host names). Second, HEARTBEAT chunks are employed to monitor peers and path status (active or inactive), in configurable intervals. SCTP uses a selective acknowledgments (SACKs) mechanism to enable accurate RTT measurements over each path and fast retransmission of missing data. Finally, for path selection, as the association setup proceeds, an active path is chosen as the primary path, among the several that can be available. Moreover, applications can configure the behavior of SCTP, though the SCTP API [28]. For instance, to support connection-oriented features (e.g. as TCP) or connection-less features (e.g. as UDP).

SCTP with its innate features to support multihoming, has been extended to include support for mobility (mSCTP) [30] and to enable concurrent transfers over multiple paths (CMT) [35]. mSCTP allows dynamic address reconfiguration by modifying IP addresses that were negotiated during the SCTP association setup. Such support is specified with new message types that contain the IP address and parameters to indicate the operation to perform, namely add, remove or modify the primary address. mSCTP can be employed by fault-tolerant applications, which require fast recovery.

Concurrent Multipath Transfer (CMT) [35] adds to SCTP simultaneous data transfer capabilities across multiple paths. CMT addresses some performance issues of SCTP, such as unnecessary fast retransmission at the sender and increased ACK traffic due to fewer delayed ACKs. If the available paths have unequal delay or bandwidth, a standard SCTP receiver can experience packet reordering, which will consequently lead to fast retransmission at the

sender. CMT mitigates these issues by introducing modifications in the SCTP specification, where a receiver delays the ACKs, instead of immediately acknowledging out-of-order packets. Further, the packet loss measurement mechanism takes into consideration historical information, in addition to the information conveyed by SACKs.

SCTP is not used widely as TCP or UDP, nevertheless it is already supported in several operating systems, such as Linux, FreeBSD and Mac OS. In addition, applications requiring high availability benefit with SCTP, such as Reliable Servers Pool (RFC3237).

Applications based on SCTP rely on the protocol robustness and reliability. However, it may face some problems when the network configuration changes. Some recent works try to address the potential drawbacks caused to SCTP-aware applications due networking changes [42, 59] requiring further SCTP reconfiguration to maintain reliability and efficiency at high levels.

Within TRONE project we aim to use a probe for collecting SCTP metrics and further using them to reconfigure the protocol. This may be needed when changes on the system or network occur such as a network card operational or hardware problem and connection link intermittent faults. In these cases, proactive SCTP reconfiguration will improve the reliability and performance of SCTP-aware applications.

Remarks on related work

Cloud infrastructure monitoring tools and security information event managers can be combined to increase the capacity of collecting and analyzing performance, fault and security data (logs, events, among other things) inside today and future cloud infrastructures. This is an important way to improve intentional or non-intentional faults detection, diagnosis and correction over IaaS. These cloud infrastructures can be characterized as critical systems because they are used to provide every kind of service, ranging from normal applications to very critical services. In each case, users heavily rely on cloud providers to have their services 100% available, secure and reliable. Any security threat may impact in financial and personal losses. Depending on the service being provided, such as energy grid and health, any failure is critical and may led to a big impact for many customers.

The most common feature among all tools resides on the final goal of monitoring IT infrastructure, specially computing clouds. Another three features could also be added, which are: a) they work all in a centralized way, having a central point of incoming data and access; b) some of them were designed for specific environments and/or scenarios; and c) they are

all vulnerable to intrusions and faults tolerant prone because their architecture does not rely on resilient replicated components. The latter could lead to critical problems like data corruption. An attacked monitoring server could, for example, compromise the overall monitoring information, which is the most important thing to be carried out in a trustworthy way in these kinds of systems because customers completely rely on the data provided by the monitoring systems. Reliability and trustworthiness are the main issues addressed by the TRONE project. The proposed architecture will use known protocols and technologies to guarantee secure and reliable monitoring services. Other tools may even become clients of the fault and intrusion tolerant architecture as it will be shown on the remaining chapters.

Publish-subscribe systems and Byzantine fault tolerant protocols are important components in the FIT monitor architecture. The first ones will provide a way and means to push and pull messages from probes to consoles in a trustworthy manner. And the second ones will provide the protocols and mechanisms needed to create a fault and intrusion tolerant system.

Chapter 3

System Model

In this chapter we introduce the network, synchrony and fault models that are going to be taken as assumptions for the proposed fault and intrusion tolerant monitoring system. Probes, consoles and the fault and intrusion tolerant monitoring system itself are assumed to work with the conditions stated by the models here presented.

3.1 Network model

We can have different network topologies [15] and technologies, even if we are talking about a company. The most common network topologies are bus, ring, star, extended star, hierarchical and mesh. They can be combined to compose hybrid topologies in a number of ways. Even a simple company may have different local networks, with different topologies, connected through gateways, routers, among other network devices.

One local physical network can be segmented in several virtual networks, called Virtual Local Area Networks (VLANs). VLANs are widely used in enterprise and academic environments to simplify address allocation across different administrative units [31]. They can serve for many purposes, like isolating different kinds of traffic, avoiding complete network blackouts due the occurrence of failures or attacks, provide better traffic control, isolate unreliable users from administrative and trusted ones, separate data from management traffic, among other applications. In most of cases the basic idea is to increase levels of flexibility or security.

Security is one of the most critical issues in cloud infrastructures [62]. VLANs are commonly used inside data centers to isolate different kinds of data traffic. Further uses include creating virtual networks attached to customers. However, VLANs may represent a security threat once they may be bypassed due misconfigurations or hopping attacks [49].

Virtual Private Network (VPNs) represent a way for companies to reduce costs [71]. A VPN enables LAN or VLAN interconnection through the public Internet without a typical private line or frame relay services [45], resulting in lower costs for companies. Two of the main challenges of this low cost alternative are security and reliability because the virtual network is configured and used over public infrastructures which can suffer from problems such as overloads, instabilities, attacks and temporary unavailability with different root causes.

Nowadays, LANs, VLANs and VPNs are commonly used with IP networks. They work with IP and run on transport level protocols like TCP and UDP both widely used by infrastructure and customer applications.

In the TRONE project we assume the TCP/IP model with fully connected networks. That is, all machines in the cloud infrastructure are accessible from each other. They may be in the same physical network, in different virtual networks or even in different physical locations. However, they see each other. In a physical LAN all machines naturally see their networked neighbors. Using different VLANs there will be at least one connection point, like a switch, a gateway or a router inside the same physical network. On the latter case, using VPNs or even private lines, all remote sites are connected through a virtual or physical communication channel. Figure 3.1 illustrates the use of VLANs and VPNs. A company X can have a LAN A and a second LAN B. Inside each local area network it may have different VLANs, each of them for a specific purpose. As an example, VLAN A for customer systems, VLAN B for storage systems and VLAN C for monitoring and management. And both physical LANs, and respective VLANs, are accessed through the Internet through a VPN. So, in theory, the company devices and systems are accessible both from LAN A and LAN B.

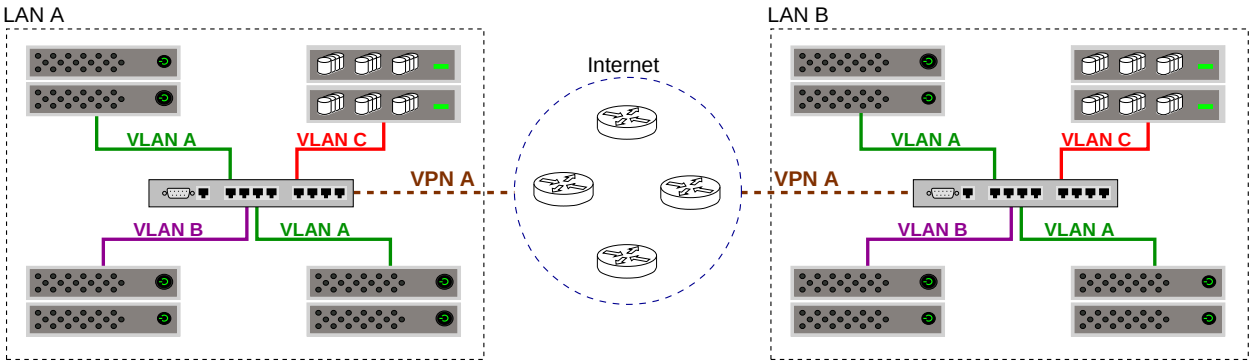

Figure 3.1: Example of enterprise networks

3.2 Synchrony model

The synchrony model refers to assumptions related to the notions of time and timeliness. Since the complexity of the solutions and the correctness of the system depend on these assumptions, they should reflect as accurately as possible the real characteristics of the execution environments. Traditionally, distributed systems have been developed by considering one of the two extreme models of synchrony. The asynchronous model, also called time-free model, does not make any time-related or timeliness assumption [29]. On the other extreme, the synchronous model assumes that all system activities are executed within known temporal bounds, which includes local activities (process execution) and distributed ones (message transmission). However, many real systems are not fully asynchronous nor fully synchronous. Therefore, there exist models of partial synchrony to cover various intermediate cases, for instance assuming that there are reliable local clocks or that only some components are temporally predictable.

In the TRONE project we focus on environments in which many computational components interact through shared networks in manners that cannot be considered predictable with respect to latency. On the other hand, current systems are equipped with very accurate and reliable hardware clocks, which are necessary for the correct system operation. These clocks are usually synchronized by means of software protocols (such as NTP [46]) and therefore it is possible to have an absolute notion of time in the overall system. This is particularly important when considering distributed operations, as we consider in TRONE.

The well-known asynchronous and synchronous distributed system models are not adequate in the case of TRONE. This model is very good in the sense that distributed algorithms are always safe, because no temporal property can be violated. However, it is impossible to reason about time or temporal relations about events, which renders the system model impractical. In the synchronous model it is possible to ensure that distributed activities are executed in bounded amounts of time, allowing the development of real-time systems. The problem is that the assumed bounds may not be secured all the time, leading to failures in distributed algorithms and systems developed over these assumptions.

It is possible to consider several models of partial synchrony [21, 23, 14, 63]. However, we believe that the most appropriate model in the case of TRONE is the timed asynchronous model [18]. This model can be described as being fundamentally an asynchronous model with the additional assumption that processes have access to a physical clock with a bounded rate of drift. This is precisely what we need. The authors of [18] have observed that most computing systems (such as the systems we consider in TRONE) have high-precisions quartz

clocks, which renders the assumption reasonable enough. Practical systems can then be built in infrastructures that alternate between synchronous and asynchronous behavior, with the system making progress when there is enough synchrony, being possible to detect timing failures otherwise.

3.3 Fault model

When characterizing service failures, from the consistency perspective, we distinguish the system failure in a) consistent failures; and b) inconsistent failures [6]. In the former case all system users get aware about incorrect service in the same way. The latter kind of system failure leads to different user views of incorrect services. Even more, some users may not notice at all service failures. These kinds of failures are known as Byzantine failures.

In TRONE we consider different fault models for the different components of the monitoring infrastructure. In fact, observing the abstractions depicted in Figure 1.4, that is, probes or consoles in the role of clients, and event brokers in the role of servers, we consider that clients are reliable and trustworthy, while servers can be affected by faults, ranging from crash to Byzantine faults. We deal with the problem of ensuring a trustworthy monitoring service in spite of Byzantine faults, which is the critical problem, because services are usually centralized. On the other hand, there are usually many probes and many consoles, which already provides a reasonable level of resilience against attacks.

So we consider all possible failures that may affect the correct behavior of such monitoring service. For instance, abnormal behavior may happen due to some power, software or hardware failure, corresponding to omission or crash failure semantics. It may also happen due to delays or corrupted data, either intentionally or accidentally inserted in the system.

Furthermore, we assume that the monitoring system can tolerate up to f faulty replicas.

Chapter 4

Fault and Intrusion Tolerant Monitoring Architecture

In this chapter, we describe the first architectural specification of the fault and intrusion tolerant monitor. We present some basic definitions for the terms used in this report. Then we explain main components. Next, we give an overview of the broker's internals. Finally, we describe also the FIT monitor's essential programming level interfaces.

4.1 Basic definitions

We present here some basic definitions that are going to be used through the first specification of the fault and intrusion tolerant monitoring architecture. These definitions will also help to better understand some essential concepts of the proposed architecture.

FIT monitor: FIT monitor represents the fault and intrusion tolerant architecture first specified in this deliverable. It has as main components replicated **brokers**, which are responsible for receiving and sending messages to the clients (probes and consoles) in a trustworthy way. The monitor is tolerant to crash and Byzantine failures.

Probe: A probe or an agent is a FIT monitor client running in monitored devices on machines, for collecting monitoring data. On occurrence of any critical event or just to report the component status, the probe will generate event messages and send them to FIT's brokers. For the proposed architecture, probes are known as event message publishers.

Console: A console is a second kind of FIT monitor client. It is a software or a hardware component that receives event messages and presents them for network or infrastructure managers. A console can be a simple display as well as a rather complex system such as a security information event manager and a high level dashboard-based monitoring system. These clients are also known as event message subscribers.

Event message: An event message is a data unit that contains information about a monitoring or alert event triggered inside the object or system which is under continuous monitoring by probes. The event message contains the essential information of state of the computing component.

Channel: It represents the communication flow inside the brokers used by probes and consoles. A channel is identified by a TAG. Each different TAG requires a different channel. Probes publish event messages in the channels while consoles will receive those messages.

TAG: A TAG is used to name a channel, which typically serves to disseminate a certain kind of events, for which common properties must be ensured. For instance, there may be channels (identified by some TAG) for event messages related to network, storage, system usage and security threats. The TAGs are used to create different communication channels inside the brokers.

CLASS: Classes are used to define characteristics of the channels identified by TAGs. Each CLASS may specify a different quality of service (QoS). If some set of messages are urgent and need to maintain message ordering, they should go through a channel instantiated with a CLASS that specifies the required quality of service. We consider that QoS can be specified along the following three vectors: fault-tolerance, order and urgency. Based on the QoS requirements of each channel, a specific CLASS must be used for that channel. Based on the QoS requirements of each TAG, the channel and CLASS parameters are fixed.

Fault-tolerance. We assume two fault models for the monitoring system: crash and Byzantine. In the crash fault model, a broker may crash and stop while in the Byzantine fault model, a broker may behave arbitrarily. Based on these two fault models along with other QoS constrains, we propose different protocols for the monitoring system.

Order. A CLASS decides the order with which the event messages will be sent to the client.By order we mean the ordering properties of the events delivered to clients, which can range from unordered delivery, to totally ordered delivery across all clients.

Other intermediate ordering requirements, such as temporal ordering or causal ordering, might also be useful for certain types of events.

Urgency. Urgency is a CLASS property which describes the urgency of action needed for specific event messages generated by a probe. For instance, some events might be very urgent and therefore it might be useful to instantiate a channel of urgent CLASS, so that events sent through this channel will be handled and forwarded as soon as possible (which might depend on the ordering requirements).

4.2 Main components

Here we explain the building blocks of the fault and intrusion tolerant monitoring architecture. Figure 4.1 shows an abstraction of the proposed FIT monitoring system.

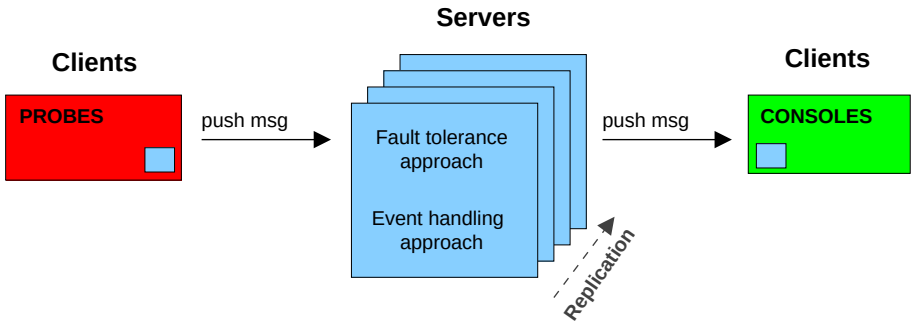

Figure 4.1: FIT monitoring system model and main components

The FIT monitoring system's main components are clients (probes and consoles) and servers (replicated event brokers). Servers are basically implemented with event handling subsystems and replication mechanisms. We explain each of these components.

Client side

The client side is composed of probes and consoles. Probes publish event messages and consoles subscribe to some channels to receive specific event messages.

Probes generate event messages on occurrence of any critical event or to report system status. They send the event messages to broker's specific channels. The event messages are then propagated by the broker to all interested consoles.

A probe can be a program residing in computing components such as network switch, router, storage system, virtual machine inside a host. It either resides in the device being monitored

or in another machine. In the latter case, it collects monitoring data through a network or other cable connection within the system being monitored.

A console is either a machine with a simple display or a more complex system with local database for storing event messages and some software to analyze, process and display statistics, alerts and critical events. The consoles are used by network operators and managers to see how the infrastructure is working and to take further decisions on the occurrence of a threat. Further, a console may even be a resource manager that automatically analyzes event messages and takes actions trying to correct or avoid problems on the monitored systems. In this case a human operator is needed only to receive the resource manager action notifications or to solve a problem that it could not automatically solve.

Consoles have also an integrated voting interface. It is needed to deal with Byzantine faults and ensure the trustworthiness of event messages. To do so, the voter mechanism performs a voting on the messages received from different broker replicas before displaying it on the console.

Probes and consoles are clients of the FIT event broker. Probes generate event messages and push them into channels. FIT event broker acts as a server receiving the event messages. It afterwards pushes the event messages to all interested consoles. Probes and consoles can either be active or passive clients. Active clients send-push information to the server while passive clients wait for the servers which are going to pull messages from them. Probes and consoles can both be active or passive and the publish-subscribe service can be implemented over TCP (connection-oriented) or UDP (connectionless).

Server side

A FIT monitor essentially provides a replicated service and acts as a server for probes and consoles. It manages all the messages received from different probes and propagates them to consoles in a trustworthy way. It provides an interface in which it receives the event messages from different probes. fault tolerance, urgency and order. The event messages of the channel will be pushed to all subscribers of that channel.

Crucial events need to be delivered to consoles despite Byzantine faults affecting the event broker service. The system administrator specifies the fault tolerance requirement for event messages. Inside the server, messages requiring only crash fault tolerance are sent to subscribers using some simple crash fault tolerant protocol with or without ordering. On the

other hand, messages requiring Byzantine fault-tolerance are forwarded using Byzantine fault tolerant protocols.

Some event messages carry crucial information and need guaranteed delivery to the console while other messages just need to be sent with best effort. There are two fault tolerant mechanism to deal with this: Byzantine and crash. The system administrator specifies the fault tolerance requirement for event messages. Inside the server, messages requiring the best effort delivery are forwarded using crash fault tolerant protocol. On the other hand, for event messages requiring guaranteed delivery are forwarded using Byzantine fault protocol. There is a trade off of choosing any one of these protocols. Although crash fault tolerant does not provide guaranteed of the event messages to consoles, it performs less time consuming operations. While BFT protocol ensures delivery of messages, it is more complex and time consuming because of the extra operations needed for agreement and ordering. In addition to fault-tolerance, other aspects of forwarding messages are ordering and urgency.

After the order of event messages to be delivered to subscribers is decided by the broker, event messages are put into queues and then sent to the respective interested subscribers/consoles.

Fault and intrusion tolerance. A FIT monitor acts as a broker and manages all monitoring event messages. However, it can be a single point of failure or attack. To continue providing trustworthy and resilient monitoring service, we replicate the event broker. We propose to use state machine replication which allows to perform arbitrary operations provided the operations are deterministic. We assume that any of these event brokers can fail or behave arbitrarily due to some malicious attack or failure. As processing of event messages is done at consoles, preserving the integrity of their content is also crucial. Towards ensuring faults and intrusions tolerance, we also propose a probe to use to use public-key cryptographic methods such as those provided by RSA to sign the event messages using its private-key. A probe can have several identities and it may impersonate other probes. Thus, a probe is required to sign event messages generated by itself, using its private key so that the broker can authenticate the probe's identity. This also ensures the integrity of the event messages generated by the probe. Since the message is being processed at the console, such cryptographic methods enable the console to verify the integrity of the event message sent by the probe.

4.3 Broker architecture

Before sending any monitoring data to the broker, probes need to see if there are channels where the event messages can be published. Figure 4.2 is an example of communication

channels inside a broker. The channels are created by the probes. Each probe can create one or more channels, with the method `Create_a_Channel(CLASS,TAG)`, for different kinds of event messages. One or more probes can register to publish events in the same channel, using the method `Register_to_Channel(TAG)`. As can be seen in the figure, two probes get registered for channel T1. After the registration, both probes will be able to send event messages to that channel. The broker will maintain a registration table consisting of probes and the corresponding channels they are assigned to deliver monitoring data. We assume that each channel can support a pre-defined number of registered probes. This limit is defined based on the broker processing capacity. So, it is going to be specific to each deployment environment.

A console can subscribe to one or more channels using the `Subscribe_to_Channel(TAG)` method. Once it is subscribed, the console will start to receive every event message that is propagated through the channel. The broker maintains a subscriber table to identify each console with the corresponding channels. This information is used to deliver the event messages to the right subscribers. The number of consoles subscribed to one or more channels is limited by the broker processing capacity, similarly to the case of the probes.

There can be a channel which is not being used for a long period of time, neither by publishers nor by subscribers. The channel T4, in the Figure 4.2, represents a communication path not being used either by probes or subscribers. Such channel can be removed from the system. For each channel there will be a timeout value to decide if the channel should be kept or not in the system. If the timeout value expires and the channel did not receive any event message during the period of time, it can be removed from the system. If a probe related with the removed channel was down and comes up again, it can create once again the same channel. All consoles subscribed to the channel before its removal will be notified and are able to subscribe again.

A broker manages every event message that arrives from probes. To deal with event messages arrived from probes and to deliver them to respective subscribers, the following broker components are important: interface, Crash Fault Tolerant (CFT) protocol, Byzantine Fault Tolerant(BFT) protocol and TAG-based filter. Details of the broker's architecture with crash and Byzantine fault tolerant protocol are given in Figures 4.3 and 4.4, respectively.

Interface. The interface provides five methods to enable communication flows between publishers, brokers and subscribers. It allows creation of an event channel between a publisher and a broker, probes to register and publish messages to a channel, and subscribers to subscribe and receive messages to/from a channel. The channels created inside a broker are input queues that receives event messages. Through the interface, the channel charac-

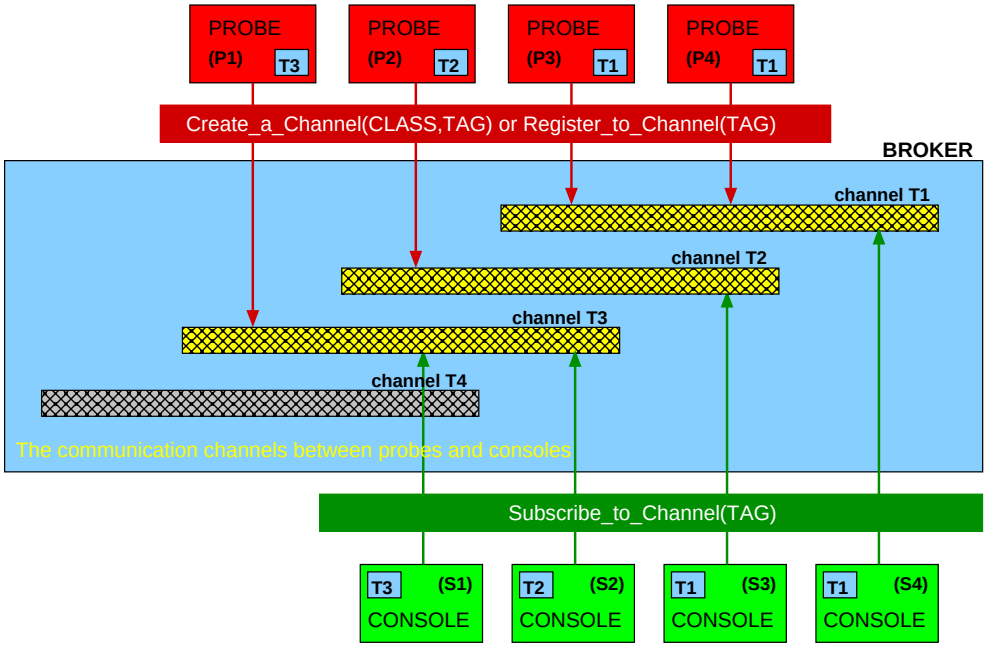

Figure 4.2: Dealing with channels inside the broker

teristics is defined. For dealing with the channels the broker will have a table with columns $< CLASS, TAG >$. It will be filled with entries stating different TAGs and associated with different CLASSes specifying QoS requirements. The interface is the same for both crash and Byzantine fault tolerant protocols.

CFT protocol. The CFT protocol is used inside the broker when an event message requires to reach subscribers in presence of crash faults in the channels. Some of these event messages may carry time-critical event information requiring urgent treatment while others may not carry time-critical information but may carry crucial information. There can be other messages that may not be critical. Some of theses messages may or may not require ordering. Within CFT protocol, based on the QoS specified by the channel, different actions are taken.There can be several sub-protocols to deal with various combinations of the quality of services. Here, we are considering three specifications of the CLASSes for CFT protocol: CFT_O, CFT_NO_U and CFT_NO_NU. The CFT protocol has two sub-protocols: CFT with order (CFT_O) and CFT without order (CFT_NO). The CFT protocol uses CFT_NO to deals with even messages for the CLASSes specified by CFT_NO_U and CFT_NO_NU. In CFT_NO, event messages that need urgent treatment are put in an urgent (U) queue while others are put in a not urgent(NU) queue. Based on a dynamic scheduling policy, the event messages from urgent and not-urgent queue are forwarded to the TAG-based filter.

Figure 4.3 shows the data flows inside the broker with crash fault tolerant protocols. As can

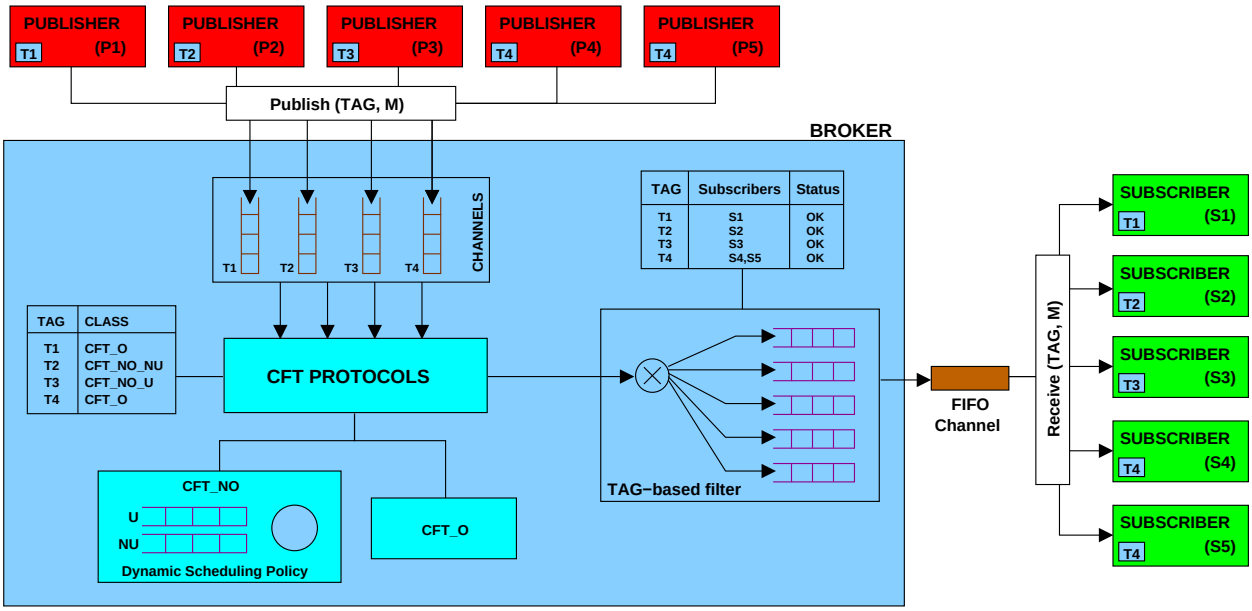

Figure 4.3: Broker architecture with crash fault tolerance

be seen in the figure, there are five probes {P1, P2, P3, P4, P5} publishing event messages to different channels, which are identified by four TAGs {T1, T2, T3, T4}. It should be also noted that there are two probes {P4, P5} publishing messages to the same channel T4. The channels inside the brokers are represented by queues that receive the event messages and forward it to the CFT protocol. Based on the QoS requirements of each channel, different CFT protocols can be used. The TAG-based filter receives event messages from CFT protocols and, based on the TAG on event messages, fills the output queue of each subscriber. To do so, it uses the subscription table for filtering the event messages. Each subscriber receives event messages sent through a FIFO channel. As shown in the figure, there are five subscribers {S1, S2, S3, S4, S5}, subscribing to different channels. Subscribers {S1, S2, S3} receives event messages with TAGs {T1, T2, T3}, respectively while subscribers {S4, S5} receives event messages with same TAG T4.

BFT protocol. The broker uses BFT protocol to maintain total order between the event messages requiring to reach subscribers in presence of Byzantine faults across the channels. There is only one CLASS, BFT, for specifying the fault tolerance requirement. Since, any BFT algorithm maintains a total order, the ordering requirement is implicit.

The BFT protocol uses state machine replication to tolerate Byzantine failures. Each broker behaves as a state machine replica. The event messages across different channels requiring Byzantine fault tolerance becomes inputs to the state machine replica. To decide on the

order of event messages, brokers use an agreement protocol among themselves. Once the ordering of the event messages is decided, it is put in an abstract queue before forwarding it to the TAG-based filter. The TAG-based filter puts event messages in a separate queue specific to a subscriber based on the TAG it has subscribed. The order in which the messages are put in the queue specific for a subscriber denotes the state of the state machine replica. Any order other than the decided one at any broker will lead the broker to a different state as the TAG-based filter will then forward event messages out of order to the queue specific for a subscriber which in turn reach subscribers in out of order. The queue before TAG-based filter contains event messages from different channels for different subscribers. After TAG-based filter, there is separate queue for each subscriber. Since the processing is done at the subscribers, the output of all state machine replicas must reach subscribers in the same order. This is required when different subscribers subscribed to the same system status information or critical events. After ordering, stored in the queue for a subscriber based on the TAG, the event messages are sent to the respective subscribers assuming a FIFO channel between TAG-based filter and the subscriber.

The BFT protocol also requires to run the agreement protocol to decide on the subscription table entries. When a subscriber requests for subscription to a specific TAG, all correct brokers must receive the request and agree on updating the table so that they will have same copy of the table. This is essential because TAG-based filter uses this table to filter event messages based on TAG and puts it in the subscriber's queue. So, brokers must agree on the list of subscribers interested on a specific TAG. Thus, the subscription table also becomes the part of the state of the state machine replica.

Figure 4.4 shows the data flows inside the broker with Byzantine fault tolerant protocols, using state machine replication model. As stated before for the crash fault tolerant protocol, there are five publishers and five subscribers, using different communication channels. The main difference between CFT and BFT is related with the state of the subscription table and the TAG-based filter output queues. When using BFT-SMR all replicas are going to execute every command in the same order, generating the exact same output. This requires an agreement and total ordering protocols to guarantee the same state among all replicas.

TAG-based filter. The TAG-based filter contains a subscription table with columns $< TAG, Subscibers, Status >$. It uses the table to filter event messages based on their TAGs and fills the output queue specific to a subscriber accordingly. In the case of BFT protocol, these output queues for sending event messages to specific subscriber also becomes the part of the states in the state machine replica.

Voter At the subscriber, there is a voter interface that deals with trustworthiness of the

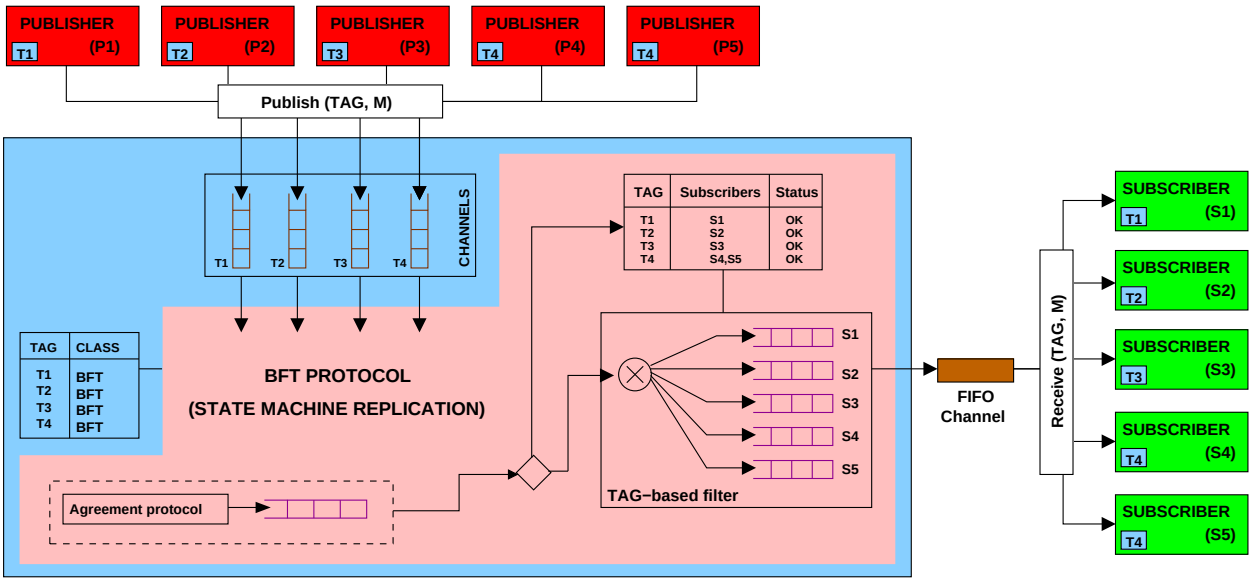

Figure 4.4: Broker architecture with Byzantine fault tolerance

event messages. It is used in conjunction with the BFT protocol. It collects event messages sent by broker's TAG-based filter and does a majority voting before delivering them to applications such as displays. At the voter, there will be a table with entries for $< probe_id, SEQ, B_1 ... B_m >$. The voter fills this table on receiving event messages from the broker and does a majority operation for each row.

4.4 Common interfaces

The broker provides four interface methods:

1. `Create_a_Channel(CLASS,TAG)`: An event channel is created based on a TAG between a broker and a publisher. A subscriber interested in a specific TAG subscribes to respective TAG and receives data from the broker corresponding to it. A channel is created for each TAG. A TAG may be associated with several publishers and brokers. A TAG in the monitoring system can be a state of processor, disk usage, among other things.

   A CLASS denotes how event messages in the channel will be treated or propagated. It specifies the characteristics of the channel. A CLASS may incorporate characteristics such as Ordering, Urgency and Fault-tolerance.

   The channel creation method is called during the bootstrapping of the monitoring system or by any management software.

2. `Register_to_Channel(TAG)`: A probe requiring to publish messages for the same channel will do a registration to the channel using this method.

3. `Publish(TAG, M)`: The probe publishes an event message by calling `Publish(TAG, M)` method. TAG is the topic that identifies to which channel the message belongs. A TAG is associated with channel. The channel has a quality of service CLASS associated. Therefore, CLASS is not required in the `publish()` method.

   M is a event message that is generated by a probe and it has the following format.

$$M =< probe\_id, SEQ, DATA, t_s >_{\sigma_{probe\_id}}$$

   where

   - *prob_id* denotes the identification number of the probe. Each probe must have an unique identifier which helps in identifying probe generating a specific message.
   - SEQ is the sequence number of the event message generated by the probe with *prob_id*
   - $DATA$ is the composition of attributes and values being monitored in the systems.
   - t_s denotes time stamp when the message is generated.
   - $< m >_{\sigma_{prob\_id}}$ denotes the message digitally signed by the Probe with its private key σ_{prob_id}.

4. `Subscribe_to_Channel(TAG)`: Each subscriber will subscribe to a specific TAG by calling the method Subscribe(TAG). Thus, a subscriber knows about the channel associated with a TAG. Several subscribers can be associated with a single TAG.

5. `Receive(M, TAG)`: This method is provided by client's interface. A subscriber or client receives the event message in the channel associated with TAG.

Chapter 5

TRONE-aware Applications

5.1 SCTP-based applications

One of the TRONE-aware application will be a SCTP control agent. It will work as a probe, collecting SCTP metrics and sending them to FIT monitor, and also as a console, receiving back control commands to reconfigure SCTP locally. The basic idea is to improve reliability and performance of local SCTP-aware applications in cases of network reconfiguration or faults. In some cases, as stated by recent works [42, 59], it is interesting and useful to proactively reconfigure SCTP protocol to avoid applications' performance and operation degradation.

The SCTP TRONE-aware application is characterized by the following capabilities:

- configure SCTP through SCTP API;
- retrieve status information of SCTP operations;
- output SCTP information;
- receive information that can enhance SCTP behavior, avoiding operation and performance degradation.

Figure 5.1 highlights the architecture of the SCTP TRONE-aware application. As can be seen, we have one publisher and one subscriber both on SCTP-aware hosts and central site where SCTP collected metrics are stored. Those collected metrics can be used by the SCTP event analyzer to identity network changes, problems and faults at hosts and then proceed the publication of SCTP reconfiguration parameters.

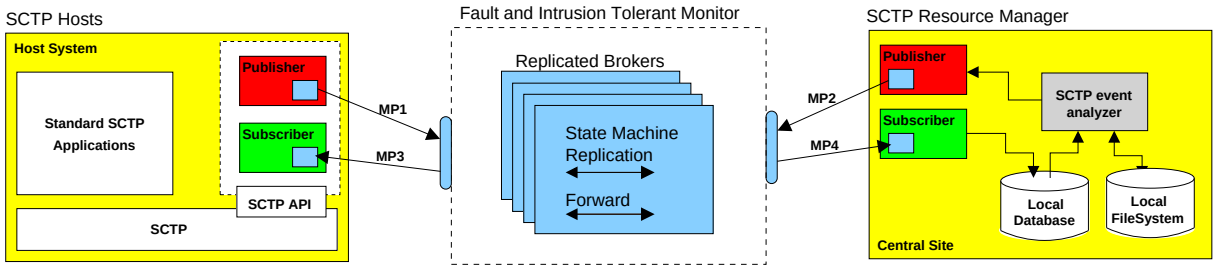

Figure 5.1: Example of TRONE-aware application within SCTP protocol

The communication channel MP1 allows the application probe to output the status of the SCTP operation. These status messages are received through communication channel MP4 and stored on a central database by a subscriber. This data is processed by a SCTP event analyzer to identify problems caused by failures or attacks.

The communication channel MP3 receives information, send by the SCTP event analyzer through a publisher using communication channel MP2, that is relevant to configure SCTP in the case of failures or other attack-related events. The collection of data is executed per request, based on a certain frequency, or an event-basis. For instance, when new associations are created, or if primary addresses are changed (e.g. can act as a pointer regarding failures in current paths). It should be noted that any possible configuration of SCTP, through the TRONE-aware application may also impact standard SCTP applications. In any case, if this is correctly managed, it can bring advantages in the event of failures.

Communication channel MP1

The information that can be out putted from the SCTP TRONE-aware application can be divided into two classes: SCTP information and implementation-specific information. With the former class, probes have information regarding the SCTP operation. With the latter, specific mechanisms can be implemented, such as to measure packet loss, delay or other relevant metrics. Table 5.1 summarizes the SCTP information that Trone-aware application can output.

The implementation-specific information includes network performance metrics such as packet loss, delay, and delay-variation. These metrics are defined and measured according to the IP Performance Metrics [24] working group recommendations. Despite the plurality of metrics available by this group, only a subset is interesting in the context of Trone, as summarized in Table 5.2.

SCTP	Option Description
SCTP_NODELAY	Activation or de-activation of Naggle-algorithm (if not activate SCTP puts more packets in the network).
SO_RCVBUF	Receiver buffer size.
SO_SNDBUF	Sender buffer size.
SO_LINGER	To perform an abort primitive.
SCTP_PRIMARY_ADDR	Set, get the peer primary address.
SCTP_DISABLE_FRAGMENTS	If enabled no SCTP fragmentation is performed.
SCTP_FRAGMENT_INTERLEAVE	How presentation of message occur to receiver, according to three levels.
SCTP_SET_PEER_PRIMARY_ADDR	Set the primary Address of an association.
SCTP_INIT_MAXSTREAMS	Initial maximum number of streams required.
SCTP_EXPLICIT_COMPLETE	Enables or disables explicit message completion.

Table 5.1: SCTP options in TRONE-aware application

Metric	How it is measured	Description
Temporal Connectivity	RFC 2678	Measure connectivity between hosts.
Path Capacity	RFC 5136	Smallest capacity of a path on a link.
Round Trip Delay	RFC 2681	Round-trip delay across paths.

Table 5.2: IPPM metrics

5.2 Security log analyzer

Security log analyzers work with a central model. This means that they have to receive data from different sources or probes, store it locally for further analysis. With the proposed FIT monitoring system those kind of tools may become clients of the fault and intrusion tolerant monitoring system, being able to receive event messages from probes and send event messages to consoles in a trustworthy and reliable way. Doing so, an IaaS could more easily have different security log analyzers, hassle free, running on their environment because they all can use the resources and data provided by our general purpose FIT monitoring system.

In the TRONE project we want to adapt at least one security log analyzer to become a FIT monitor client. Examples could include OSSI, SALSA, Puslo or even ArchSight. Select which one is going to be used depends on technical details such as the access to its source code or an API that allows to implement extension for the tool.

5.3 Monitoring system

Monitoring systems may be FIT monitoring system clients as well. They may subscribe to receive event messages of any kind. And they also act as probes for delivering alerts and other critical messages to networks operators through the consoles.

Our idea is to analyze and adapt one monitoring tool to be a FIT monitoring system client. It will also gives us a more precise technical details over what is needed to adapt such tools to be a client of our system.

Remarks related to TRONE-aware applications

The proposed TRONE-aware applications will serve as test and validation cases. They represent three different scenarios or applications that can benefit from the FIT monitoring system.

Chapter 6

Conclusion and Next Steps

We proposed a FIT monitoring system framework which is resilient to faults and intrusions, providing trustworthiness to the network monitoring and control infrastructure. It is designed to tolerate both crash and Byzantine fault models. The framework also allows different applications such as security analyzer tools and monitoring systems to become clients of FIT monitor.

We mapped the FIT monitoring service into a topic-based publish-subscribe system. To support fault and intrusion tolerance we propose the use of BFT libraries such as SMaRt.

We believe that this first architecture specification of the fault and intrusion tolerant monitoring system will improve monitoring systems for cloud infrastructures. Monitoring and control event messages will by sent and received in a reliable and secure way. Network operators and manages will not need to worry anymore about the trustworthiness and availability of the monitoring systems.

Our next step is to validate the proposed solution in our cloud infrastructure. We are going to use probes to monitor IaaS such as OpenStack and OpenNebula and consoles to receive and display event messages sent by probes. Crash failures and attacks will be injected into the FIT monitoring system, in order to evaluate its effective resilience and trustworthiness. The results will be of particular interest to PT which have a special interest in improving the reliability of their monitoring systems.

Bibliography

[1] AlienVault. Ossim - the open source siem, 2011. www.ossim.net.

[2] Alysson Neves Bessani, et. al. bft-smart - high-performance byzantine-fault-tolerant state machine replication, 2011. http://code.google.com/p/bft-smart/.

[3] John Grundy Amani S. Ibrahim, James Hamlyn-Harris and Mohemed Almorsy. Cloudsec: a security monitoring appliance for virtual machines in the iaas cloud model. In *Proceedings of the 5th International Conference on Network and System Security*. IEEE, September 2011. http://anss.org.au/nss2011/.

[4] Amazon. Amazon cloudwatch, 2011. http://aws.amazon.com/cloudwatch/.

[5] App-Security.org. Salsa: Scalable and agile lifecycle security for applications, 2011. http://app-security.org/.

[6] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1:11–33, January 2004.

[7] Roberto Baldoni, Leonardo Querzoni, and Antonino Virgillito. Distributed event routing in publish/subscribe communication systems: a survey. Technical report, Informatic and Systems Department of Rome University, 2005. http://goo.gl/uIm2U.

[8] Alysson Bessani. From byzantine fault tolerance to intrusion tolerance (a position paper). *5th Workshop on Recent Advances in Intrusion-Tolerant Systems (WRAITS), DSN 2011: The 41th International IEEE/IFIP Conference on Dependable Systems and Networks*, 2011.

[9] BIND9.NET / BIND9.ORG. The domain name system, 2011. http://www.bind9.net/.

[10] Ł. Budzisz, R. Ferrús, A. Brunstrom, K. J. Grinnemo, R. Fracchia, G. Galante, and F. Casadevall. Towards transport-layer mobility: Evolution of sctp multihoming. *Comput. Commun.*, 31:980–998, March 2008.

[11] J. Case, M. Fedor, M. Schoffstall, and J. Davin. Ietf rfc - a simple network management protocol (snmp), 2011. `http://www.ietf.org/rfc/rfc1157.txt`.

[12] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20:398–461, November 2002.

[13] CFEngine AS. Cfengine, 2011. `http://cfengine.com/`.

[14] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 325–340, Montreal, Quebec, Canada, August 1991. ACM.

[15] D.D. Clark, K.T. Pogran, and D.P. Reed. An introduction to local area networks. *Proceedings of the IEEE*, 66(11):1497 – 1517, nov. 1978.

[16] Cloudkick. Cloudkick - cloud management, 2011. `https://www.cloudkick.com`.

[17] CorreLog, Inc. Correlog: High performance correlation, search and log management, 2011. `http://correlog.com/`.

[18] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10:642–657, June 1999.

[19] Logan Welliver Dan Di Spaltro, Alex Polvi. Methods and systems for cloud computing management, 06 2011. `http://www.faqs.org/patents/app/20110131335`.

[20] António Manuel de Carvalho Alegria. Verificao automtica de modelos de arquitectura tecnolgica de sistemas de informao em rede, 2009. `https://dspace.ist.utl.pt/bitstream/2295/569237/1/dissertacao.pdf`.

[21] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34:77–97, January 1987.

[22] R. Droms. Ietf rfc - dynamic host configuration protocol, 1997. `http://www.ietf.org/rfc/rfc2131.txt`.

[23] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[24] Wesley Eddy and David Harrington. Ippm status pages - ip performance metrics, 2011. http://tools.ietf.org/wg/ippm/.

[25] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Survey*, 35:114–131, June 2003.

[26] Françoise Fabret, H. Arno Jacobsen, François Llirbat, João Pereira, Kenneth A. Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. *SIGMOD Rec.*, 30:115–126, May 2001.

[27] FeedZai. Feedzai pulse, 2011. http://www.feedzai.com/.

[28] G. Fekete. Network interface management in mobile and multihomed nodes, 2010. http://dissertations.jyu.fi/studcomp/9789513939236.pdf.

[29] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the Association for Computing Machinery*, 32(2):374–382, April 1985.

[30] John Fitzpatrick, Séan Murphy, Mohammed Atiquzzaman, and John Murphy. Using cross-layer metrics to improve the performance of end-to-end handover mechanisms. *Comput. Commun.*, 32:1600–1612, September 2009.

[31] Prashant Garimella, Yu-Wei Eric Sung, Nan Zhang, and Sanjay Rao. Characterizing vlan usage in an operational network. In *Proceedings of the 2007 SIGCOMM workshop on Internet network management*, INM '07, pages 305–306, New York, NY, USA, 2007. ACM.

[32] Xiangfeng Guo, Jun Wei, and Dongli Han. Efficient event matching in publish/subscribe: Based on routing destination and matching history. *Networking, Architecture, and Storage, International Conference on*, 0:129–136, 2008.

[33] HP. Arcsight. http://www.arcsight.com/products/products-esm/.

[34] IBM. Tivoli security information and event manager, 2011. http://www-01.ibm.com/software/tivoli/products/security-info-event-mgr/.

[35] Janardhan R. Iyengar, Paul D. Amer, and Randall Stewart. Concurrent multipath transfer using sctp multihoming over independent end-to-end paths. *IEEE/ACM Trans. Netw.*, 14:951–964, October 2006.

[36] Don Jones. *Creating Unified IT Monitoring and Management in Your Environment.* Realtime publishers, 1 edition, 2011. http://www.nimsoft.com/.

[37] Reza Sherafat Kazemzadeh and Hans-Arno Jacobsen. Reliable and highly available distributed publish/subscribe service. In *Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems*, pages 41–50, 2009.

[38] J. Klensin. Ietf rfc - simple mail transfer protocol, 2001. http://www.ietf.org/rfc/rfc2821.txt.

[39] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst.*, 27:7:1–7:39, January 2010.

[40] Ying Liu and Beth Plale. Survey of publish subscribe event systems. Technical report, Computer Science Dept. of Indiana University, 2003. Technical Report number 574 available at https://www.cs.indiana.edu/l/www/ftp/techreports/TR574.pdf.

[41] logicMonitor. Logicmonitor - hosted monitoring of network, servers, applications, storage, and cloud, 2011. http://www.logicmonitor.com/.

[42] S. Maruyama, M. Kozuka, Y. Okabe, and M. Nakamura. Policy-based ip address selection in sctp automatic address reconfiguration. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pages 704 –707, march 2011.

[43] Microsoft. System center operations manager, 2011. http://www.microsoft.com/en-us/server-cloud/system-center/operations-manager.aspx.

[44] Matteo Migliavacca and Gianpaolo Cugola. Adapting publish-subscribe routing to traffic demands. In *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, DEBS '07, pages 91–96, New York, NY, USA, 2007. ACM.

[45] Stanislav Milanovic and Zoran Petrovic. Deploying ip-based virtual private network across the global corporation. In *The 1th WSEAS International Conference on Applied Computer Science*. WSEAS, 2001.

[46] D. Mills. Network time protocol (version 3) specification, implementation, 1992. http://www.rfc-ref.org/RFC-TEXTS/1305/index.html.

[47] Monitis. Monitis all-in-one monitoring platform - monitor everything, 2011. http://portal.monitis.com/.

[48] R Mugesh. Snort correlation engine, 2011. http://sourceforge.net/projects/tuxsnort/.

[49] Murat Mukhtarov, Natalia Miloslavskaya, and Alexandr Tolstoy. Network security threats and cloud infrastructure services monitoring. In *Seventh International Conference on Networking and Services*, pages 141–145. IARIA, May 2011.

[50] Nimsoft. Nimsoft monitor, 2011. http://www.nimsoft.com/solutions/nimsoft-monitor.

[51] Nimsoft. Nimsoft monitor - unified monitoring for modern it, 2011. http://www.nimsoft.com/wp-content/uploads/2011/03/nimsoft-monitor-ds.pdf.

[52] OpenLDAP Foundation. Openldap software, 2011. http://www.openldap.org/.

[53] Joo Pereira, Franoise Fabret, Franois Llirbat, and Dennis Shasha. Efficient matching for web-based publish/subscribe systems. In Peter Scheuermann and Opher Etzion, editors, *Cooperative Information Systems*, volume 1901 of *Lecture Notes in Computer Science*, pages 162–173. Springer Berlin / Heidelberg, 2000. 10.1007/10722620_17.

[54] Portugal Telecom Comunicacões. Smartcloudpt, 2011. http://www.smartcloudpt.pt/.

[55] Puppet Labs. Puppet, 2011. http://puppetlabs.com/.

[56] Qualys, Inc. Quidscor, 2011. http://quidscor.sourceforge.net/.

[57] G. Santos Veronese, M. Correia, A.N. Bessani, and Lau Cheuk Lung. Ebawa: Efficient byzantine agreement for wide-area networks. In *High-Assurance Systems Engineering (HASE), 2010 IEEE 12th International Symposium on*, pages 10 –19, nov. 2010.

[58] Dawn Xiaodong Song and Jonathan K. Millen. Secure auctions in a publish/subscribe system, 2000.

[59] R. Stewart, Q. Xie, M. Tuexen, S. Maruyama, and M. Kozuka. Ietf rfc - stream control transmission protocol (sctp) - dynamic address reconfiguration, 2007. http://tools.ietf.org/html/rfc5061.

[60] Tenable Network Security. Tenable log correlation engine, 2011. http://www.tenable.com/products/tenable-log-correlation-engine.

[61] R. Vaarandi. Sec - a lightweight event correlation tool. In *IP Operations and Management, 2002 IEEE Workshop on*, pages 111 – 115, 2002.

[62] Luis Vaquero, Luis Rodero-Merino, and Daniel Morn. Locking the sky: a survey on iaas cloud security. *Computing*, 91:93–118, 2011. 10.1007/s00607-010-0140-x.

[63] P. Veríssimo and C. Almeida. Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models. *Bulletin of the TCOS*, 7(4):35–39, Winter 1995.

[64] VMware. Vmware vfabric hyperic, 2011. `http://www.vmware.com/products/vfabric-hyperic/`.

[65] VMware. Vmware vfabric hyperic datasheet - manage application performance across physical, virtual and cloud infrastructures, 2011. `http://www.vmware.com/files/pdf/VMware-vFabric-Hyperic-DS-EN.pdf`.

[66] M. Wahl, T. Howes, and S. Kille. Ietf rfc - lightweight directory access protocol (v3), 1997. `http://www.ietf.org/rfc/rfc2251.txt`.

[67] Timothy Wood, Rahul Singh, Arun Venkataramani, Prashant Shenoy, and Emmanuel Cecchet. Zz and the art of practical bft execution. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 123–138. ACM, 2011.

[68] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, and Mike Dahlin. Separating agreement from execution for byzantine fault tolerant services. *SIGOPS Oper. Syst. Rev.*, 37:253–267, October 2003.

[69] Zenoss. Zenoss - the cloud management company, 2011. `http://www.zenoss.com/`.

[70] Zenoss. Zenoss for enterprise it, 2011. `http://mediasrc.zenoss.com/documents/wp_zenoss_for_enterprise_IT.pdf`.

[71] Zhensheng Zhang, Ya-Qin Zhang, Xiaowen Chu, and Bo Li. An overview of virtual private network (vpn): Ip vpn and optical vpn. *Photonic Network Communications*, 7:213–225, 2004. 10.1023/B:PNET.0000026887.35638.ce.