Service and Protocol Architecture for the MAFTIA Middleware

C. Cachin, M. Correia, T. McCutcheon
N. F. Neves, B. Pfitzmann, B. Randell
M. Schunter, W. Simmonds, R. Stroud
P. Verissimo, M. Waidner, I. Welch

DI–FCUL TR–01–1

January 2001

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Project IST-1999-11583

Malicious- and Accidental-Fault Tolerance for Internet Applications

Service and Protocol Architecture for the MAFTIA Middleware

Paulo Veríssimo and Nuno Ferreira Neves (editors)
University of Lisboa

MAFTIA deliverable D23

Public document

March 11, 2001

Technical Report DI/FCUL TR-01-1, University of Lisboa

Technical Report CS-TR-721, University of Newcastle upon Tyne

Editors

Paulo Veríssimo, *University of Lisboa (P)*
Nuno Ferreira Neves, *University of Lisboa (P)*

Contributors

Christian Cachin, *IBM Zurich Research Lab (CH)*
Miguel Correia, *University of Lisboa (P)*
Tom McCutcheon, *DERA, Malvern (UK)*
Nuno Ferreira Neves, *University of Lisboa (P)*
Birgit Pfitzmann, *Universität des Saarlandes (D)*
Brian Randell, *University of Newcastle upon Tyne (UK)*
Matthias Schunter, *Universität des Saarlandes (D)*
William Simmonds, *DERA, Malvern (UK)*
Robert Stroud, *University of Newcastle upon Tyne (UK)*
Paulo Veríssimo, *University of Lisboa (P)*
Michael Waidner, *IBM Zurich Research Lab (CH)*
Ian Welch, *University of Newcastle upon Tyne (UK)*

Contents

List of Figures

Abstract

This document describes the specification of the MAFTIA middleware architecture. This specification focusses on the models, building blocks and services. It describes the tradeoffs made in terms of models, the choices of building blocks and their topology, and the portfolio of services to be offered by the MAFTIA middleware to applications and high-level services. In particular, regarding the system model, it presents a detailed discussion on the fault, synchrony, topological, and group models, which were used to guide the overall architecture. The architecture was divided into two main levels, the site part which connects to the network and handles all inter-host operations, and a participant part which takes care of all distributed activities and relies on the services provided by the site-part components.

1 System Model

Paulo Veríssimo, Nuno Ferreira Neves, Miguel Correia, *University of Lisboa (P)*
Christian Cachin, *IBM Zurich Research Lab (CH)*
Brian Randell, Robert Stroud, Ian Welch, *University of Newcastle upon Tyne (UK)*

1.1 Fault Model

A crucial aspect of any architecture is the fault model upon which the system architecture is conceived, and component interactions are defined. The fault model conditions the correctness analysis, both in the value and time domains, and dictates crucial aspects of system configuration, such as the placement and choice of components, level of redundancy, types of algorithms, and so forth. There are essentially two different kinds of failure assumptions underlying a fault model: controlled failure assumptions; and arbitrary failure assumptions.

1.1.1 Failure Assumptions

Controlled failure assumptions specify qualitative and quantitative bounds on component failures. For example, the failure assumptions may specify that components only have timing failures, and that no more than f components fail during an interval of reference. Alternatively, they can admit value failures, but not allow components to spontaneously generate or forge messages, nor impersonate, collude with, or send conflicting information to other components. This approach is extremely realistic, since it represents very well how common systems work under the presence of accidental faults, failing in a benign manner most of the time. It can be extrapolated to malicious faults, by assuming that they are qualitatively and quantitatively limited. However, it is traditionally difficult to model the behaviour of a hacker, so we have a problem of coverage that does not recommend this approach unless a solution can be found.

Arbitrary failure assumptions specify no qualitative or quantitative bounds on component failures. Obviously, this should be understood in the context of a universe of "possible" failures of the concerned operation mode of the component. For example, the possible failure modes of interaction, between components of a distributed system are limited to combinations of timeliness, form, meaning, and target of those interactions (let us call them messages). In this context, an arbitrary failure means the capability of generating a message at any time, with whatever syntax and semantics (form and meaning), and sending it to anywhere in the system.

Moreover, practical systems based on arbitrary failure assumptions very often specify quantitative bounds on component failures, or at least equate tradeoffs between resilience of their solutions and the number of failures eventually produced [7]. Arbitrary failure assumptions are costly to handle, in terms of performance and complexity, and thus are not compatible with the user requirements of the vast majority of today's on-line applications.

Hybrid failure assumptions combining both kinds of failure assumptions would be desirable. Generally, they consist of allocating different assumptions to different subsets or components of the system, and have been used in a number of systems and protocols. Hybrid models allow stronger assumptions to be made about parts of the system that can justifiably be assumed to exhibit fail-controlled behaviour, whilst other parts of the system are still allowed an arbitrary behaviour. This is advantageous in modular and distributed system architectures such as MAFTIA. However, this is only feasible when the model is well-founded, that is, the behaviour of every single subset of the system can be modelled and/or enforced with high coverage, and this brings us back, at least for parts of the system, to the problem identified for controlled failure assumptions.

1.1.2 Composite Fault Model

The problems identified in our discussion of failure assumptions point to the need for the MAFTIA fault model to have characteristics enabling the definition of intermediate, hybrid assumptions, with adequate coverage. A first step in this direction is the definition of a composite fault model specifically aimed at representing the failures that may result from several classes of malicious faults. A second step is the definition of a set of techniques that act at different points within this composite fault model and which, combined in several ways, yield dependability vis-à-vis particular classes of faults. We are going to base our reasoning on two guiding principles:

- the sequence: attack + vulnerability → intrusion → failure
- the recursive use of fault tolerance and fault prevention

Concerning the mechanisms of failure, Figure 1.1 represents the fundamental sequence: *attack + vulnerability → intrusion → failure*. It distinguishes between several kinds of faults capable of contributing to a security failure. Vulnerabilities are the primordial faults existing inside the components, essentially design or configuration faults (e.g., coding faults allowing program stack overflow, files with root setuid in UNIX, naive passwords, unprotected TCP/IP ports). Attacks are malicious interaction faults that attempt to activate one or more of those vulnerabilities (e.g., port scans, email viruses, malicious

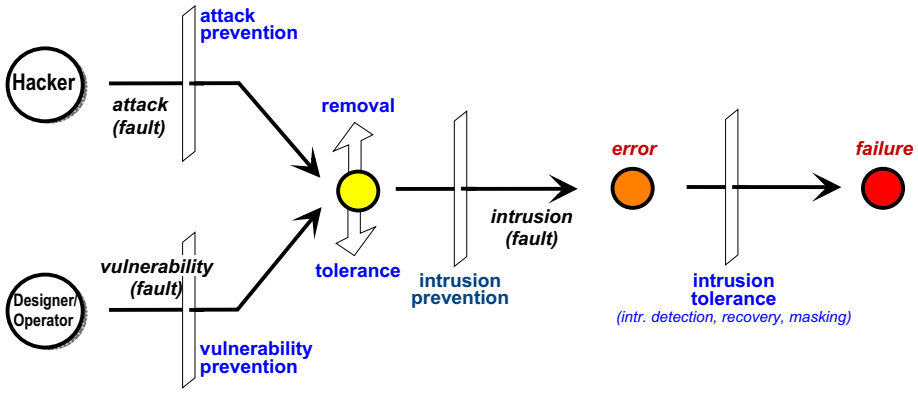

Figure 1.1: The Composite Fault Model of MAFTIA.

Java applets or ActiveX controls). An attack that successfully activates a vulnerability causes an intrusion. This further step towards failure is normally characterized by an erroneous state in the system that may take several forms (e.g., an unauthorized privileged account with telnet access, a system file with undue access permissions to the hacker). Such erroneous states can be unveiled by intrusion detection, as we will see ahead, but if nothing is done to process the errors resulting from the intrusion, failure of one or more security properties will occur.

The composite model embraced in MAFTIA allows the combined introduction of several techniques. Note that two causes concur to create an intrusion, as shown in Figure 1.1: vulnerabilities and attacks.

To begin with, we can prevent some attacks from occurring, thereby reducing the level of threat imposed on the system. Attack prevention can be performed, for example, by selectively filtering access to parts of the system (e.g., if a component is behind a firewall and cannot be accessed from the Internet, it cannot be attacked from there). However, it is impossible to prevent all attacks (e.g., some components have to be placed outside the firewall in a Demilitarised Zone), and in consequence, other measures must be taken.

On the vulnerability side, vulnerability prevention helps to reduce the degree of vulnerability by construction. However, many systems are assembled from COTS components that contain known vulnerabilities. When it is not possible to prevent the attack(s) that would activate these vulnerabilities, a first step would be to attempt vulnerability removal. Sometimes this is done at the cost of eliminating the system functions that contain the vulnerabilities.

The above-mentioned approaches can be complemented, still at the attack level, with tolerance measures achieved by combinations of the classic techniques: detection, recovery, and masking. The detection of port scans or other anomalous activity at the external border of the system forms part of the functionality of some systems generically

known as Intrusion Detection Systems (IDS). Although an intrusion has not yet occurred, there is an erroneous symptom that can be addressed by several attack countermeasures. For example, honey-pots and other evasive measures at the periphery of the system can be used to mask the effects of the attack.

The various combinations of techniques discussed above provide a range of alternatives for achieving intrusion prevention (see Figure 1.1), i.e. attempting to avoid the occurrence of intrusions. Whilst this is a valid and widely used approach, its absolute success cannot be guaranteed in all situations, and for all systems. The reason is obvious: it may not be possible to handle all attacks, either because not all attacks are known or new ones may appear, or because not all attacks can be guaranteed to be detected or masked. Similar reasoning applies to vulnerabilities. In consequence, some attacks will succeed in producing intrusions, requiring forms of intrusion tolerance, as shown in the right part of Figure 1.1, in order to prevent system failure. Again, these can assume several forms: detection (e.g., of intruded account activity, of Trojan horse activity); recovery (e.g., interception and neutralization of intruder activity); or masking (e.g., voting between several components, including a minority of intruded ones) [34].

The above discussion has laid the foundations for achieving our objective: a well-founded hybrid fault model, that is, one where different components have different faulty behaviours. Consider a component for which a given controlled failure assumption was made. How can we achieve coverage of such an assumption, given the unpredictability of attacks and the elusiveness of vulnerabilities? The key is in a recursive use of fault tolerance and fault prevention. Think of the component as a system: it can be constructed through the combined use of removal of internal vulnerabilities, prevention of some attacks, and implementation of intrusion tolerance mechanisms internal to the component, in order to prevent the component from exhibiting failures.

Looked upon from the outside now, at the next higher level of abstraction, the level of the outer system, the would-be component failures we prevented restrict the system faults the component can produce. In fact we have performed fault prevention, that is, we have a component with a controlled behaviour vis--vis malicious faults. This principle:

- establishes a divide-and-conquer strategy for building modular fault-tolerant systems;
- can be applied in different ways to any component;
- can be applied recursively at as many levels of abstraction as are found to be useful.

Components exhibit a coverage that is justifiably given by the techniques used in their implementation, and can subsequently be used in the construction of fault-tolerant protocols under the hybrid fault model.

Since we are dealing with systems of interacting components we should also address error confinement. Errors can propagate from one component to another when messages are sent from one component to another component. One approach to confining errors is to replicate components, and apply a message selection protocol that validates the messages generated by the replicated components before they are propagated. The selection protocol cross-checks each message against equivalent data messages generated by the other replicas and only sends messages that the majority of replicas agree upon. Thus, messages are only propagated if the replicas reach a consensus about them. The drawback of this *validate-before-propagate* [76] approach is that every replica must agree to send a message, so propagation is limited by the slowest replica in the group. A more efficient approach is *propagate-before-validate* [76]. With this approach the first replica to generate a message propagates the message to other components without any validation taking place. However, at some later point the computation being carried out by the components is suspended until all previous computation is validated. One plausible implementation approach would be to use a transactional framework. All interacting components would join a transaction and before commitment could take place, all message sending would be suspended and all messages sent during the transaction would be validated by cross-checking the messages generated by component replicas. If any of the messages were invalid then the transaction would be aborted and rollback would take place, otherwise the transaction would be committed and the effects of the message interchange made permanent.

This approach was originally developed for components that were active replicas. We would like to apply the concept to general components. However, standard transactional frameworks are unsuitable for this as they support only backward error recovery and a general component may not support rollback. Therefore the framework used for error confinement should support forward recovery techniques as well as backward recovery techniques. The coordinated atomic actions (CA actions) concept [106] [107] may offer a suitable framework. CA actions are intended for cooperating or competing general components involved in a joint interaction and impose strong controls on error confinement and error recovery activities in the event of failure. The implementation of the CA action mechanism would need to be made both fault tolerant and secure to be used as a framework for error confinement in a intrusion tolerant system. Therefore, as part of the MAFTIA project we intend to investigate the usefulness of an extended concept of CA actions for error confinement.

1.1.3 Classes of Threats

In an adversarial environment, making assumptions about the presumed behavior of an opponent is difficult. The failure assumptions developed in the previous sections essentially determine the general form of the attacks that we want to defend against. In

other words, we can say they determine which actions of the opponent are considered legal. But beyond that, there is not much that can be said in a security context, since a malicious attacker may invest more resources into an attack than presumed, and/or use its power in arbitrary ways.

The approach taken in security engineering and cryptography is that an adversary always knows the complete design and specification of the system, with the sole exception of the secret keys. This prudent principle, known as *Kerckhoff's assumption*, has proven over and over to be an extremely reasonable way to design cryptographically secure systems. The history of cryptography is littered with broken systems whose security rested on their design specifications remaining confidential (so-called *security by obscurity*). They typically were broken soon after some outsider had access to the algorithms—the GSM encryption algorithm and the DVD "content scrambling system" algorithm are only the most recent examples. We shall follow Kerckhoff's assumption as a general design principle.

Several established methods for enhancing fault-tolerance rely on replication to mask faulty parties. For a homogeneous distributed system, imagine that there exist n servers, of which up to t may fail in arbitrary ways. We assume that there is a single, global adversary that has corrupted up to t parties. They may exhibit arbitrary behavior, so we cannot rely on them in any form. For simplicity, we typically absorb them into the adversary and do not regard them as part of the system at all. The remaining parties are called honest. Any statement about the common state of the system can only rely on the honest parties.

Since the network is insecure, we may also assume that the protocol execution is defined entirely by the adversary within the imposed timing bounds. The adversary has arbitrary computational power within the limits of what is considered to be reasonable (technically speaking, it must be modeled as a polynomial-time Turing machine) and the boundaries imposed by the system model. Thus, the adversary controls the network, delivers messages at its discretion, and schedules the actions of honest parties. Protocols formulated in this model are not guaranteed to work unconditionally, but only to the extent that the adversary delivers messages between honest parties. In short, *the network is the adversary*. This view also takes denial-of-service attacks into account in a feasible way.

In a security context with cryptographic secrets, the above model must be interpreted such that a faulty party is regarded as faulty for the remaining lifetime of the system, or at least until the system reconfigures such that its cryptographic secrets are no longer useful to the adversary. A simple recovery procedure is not effective, in particular, when threshold-cryptographic protocols are used (see also Section 2.2.2). A malicious intruder cannot be forced to forget something he has seen; in a fault tolerance context, the solution is to render this knowledge useless to him. This may rely for example on so-called *proactive security* mechanisms.

Proactive security is a method to protect threshold-cryptographic schemes against a mobile adversary that can corrupt all parties during the lifetime of the system, but never more than t at once (see [22] for a survey). Proactive protocols divide time into epochs. All parties "reshare" their cryptographic secret keys between two epochs and delete all old key material. The model assumes an external mechanism for detecting corruptions and "cleaning up" a party. Because all secrets that the adversary has seen in the past become useless by resharing, the adversary never knows enough secret information to compromise the whole system.

1.1.4 Classes of Vulnerabilities

A thorough understanding of the vulnerabilities in a distributed system will help to identify and prevent or detect attacks capable of exploiting them. This is the main focus of intrusion detection research, performed in WP3 in MAFTIA. In this section we give a short classification of attacks and vulnerabilities and refer to [2] and to the respective deliverables in WP3 for further information.

One can distinguish the attack types according to three parameters. Firstly, we identify the *attacked object*, then the *attack point* through which the object is attacked and thirdly, the *attack principle* used by the adversary to attack the object. In principle, this classification should enable us to recognize potential vulnerabilities. It can also serve the evaluation of intrusion-detection systems.

A distributed computing system consists of many objects. Our classification is limited to a rather high level of granularity. On the level of a single system, the relevant objects include storage and file systems, I/O devices (network interface, keyboard), volatile memory, operating system, and processes. On the network level, the objects include the transport medium, directory and name servers, routers, firewalls, proxies, and more.

The second parameter is used to denote the point through which a given object is attacked. These are normally the interfaces provided for the intended usage of an object. Thus, for system objects, these are kernel modules, system calls, library calls, environment settings, user interface abstractions, etc. On the network level, basically any layer in a protocol stack offers interfaces that can be attacked.

The third parameter, the attack principle, is particularly important for designing intrusion-detection systems that should recognize attacks by their typical characteristics. Typical principles are bypassing access control, fooling authentication by exploiting other vulnerabilities, or creating, deleting, and modifying objects to change the behavior of the system.

1.1.5 Adversary Structures

One common approach in fault-tolerant distributed systems is to mask faults by replication. Usually it is assumed that at most a certain fraction in a set of servers may fail. This model is based on the assumption that faults occur independently of each other and affect all servers equally likely. For random and uncorrelated faults within a system, as well as for isolated external events, this seems adequate.

However, faults that represent malicious acts of an adversary may not always match these assumptions. This causes a conceptual obstacle for using the replication-based approach to achieve security in adversarial environments. In our setting, for example, if all servers in the system have a common vulnerability that permits a successful attack by an intruder, the integrity of the whole system may be violated easily. The independence assumption applies here only to the extent that the work needed for breaking into a server increases linearly with the number of machines that have to broken for a successful intrusion. With the sophisticated tools, automated exploit scripts, and large-scale coordinated attacks found on the Internet today, this assumption becomes increasingly difficult to justify.

Heterogeneity is a helpful paradigm, which further increases the above-mentioned notion of "difficulty to intrude". Suppose we implement a replicated server using several makes of operating system. Note that for a given vulnerability of one operating system, we can specify a bound on the number of components that can be intruded upon, by a single attack exploiting that vulnerability.

Generalized Adversary Structures

One solution for this problem, which we propose here, is to use *generalized adversary structures*. They can accommodate a strictly more general class of failures than with any weighted threshold structure. In the Byzantine model, a collection of corruptible servers is also called an *adversary structure*. Such an adversary structure specifies the subsets of parties that may be corrupted at the same time.

Let $\mathcal{P} = \{1, \ldots, n\}$ denote the index set of all parties $P_1, \ldots, P_n$. The *adversary structure* $\mathcal{A}$ is a family of subsets of $\mathcal{P}$ that specifies which parties the adversary may corrupt. $\mathcal{A}$ is monotone (i.e., $S \in \mathcal{A}$ and $T \subset S$ imply $T \in \mathcal{A}$) and uniquely determined by the corresponding maximal adversary structure $\mathcal{A}^*$ in which no subset contains another one. For the traditional threshold model of a threshold of corrupted parties, the adversary may corrupt up to t arbitrary parties. In this case, $\mathcal{A}^*$ contains all subsets of $\mathcal{P}$ with cardinality t.

Most protocols impose certain restrictions on the type of corruptions that they can tolerate. For a threshold adversary in an asynchronous model, $n > 3t$ is in general a necessary and sufficient condition. The analogous condition for protocols with a general adversary structure $\mathcal{A}$ is the so-called Q^3 *condition* [47]: no three of the sets in $\mathcal{A}$ cover $\mathcal{P}$. (Note that $n > 3t$ is a special case of this.)

The adversary structure specifies the (maximally) corruptible subsets of parties. Its complement is called the *access structure* and specifies the (minimally) qualified subsets that are needed to take some action. For example, it is used in secret sharing in cryptography [95], where it denotes the sets of parties who may reconstruct the shared secret. The access structure is usually the more important tool for the protocol designer than the adversary structure. In the example of the threshold system above, all sets of $t+1$ or more parties belong to the access structure.

Every adversary structure can also be described by a Boolean function g on n variables that represent a subset of $\mathcal{P}$; g outputs 0 on all characteristic vectors of corruptible subsets and 1 otherwise. For convenience, we describe them using arbitrary fan-in threshold gates Θ_k^n that output 1 if and only if at least k of their n inputs are 1 (note that AND and OR gates correspond to the special cases Θ_n^n and Θ_1^n). By associating subsets of $\mathcal{P}$ with their characteristic n-bit vector in the natural way, the function can be extended from Boolean values to arbitrary subsets of $\mathcal{P}$ (e.g., $g(S) = \Theta_{t+1}^n(S)$ in the threshold example).

Handling General Adversary Structures

Most threshold cryptography and agreement protocols traditionally found in the literature can be extended to a generalized Q^3 adversary structure $\mathcal{A}$, for which the corresponding secret sharing access structure can be implemented by a linear secret sharing scheme. This follows from the results of Cramer, Damgård, and Maurer [31], who show how to construct cryptographically secure schemes for general operations on secret values.

The existence of a suitable linear secret sharing scheme is important, and we will have to describe suitable secret sharing schemes together with the protocols. For example, it seems reasonable to differentiate between groups of servers who share the same values of a particular attribute, such as location or operations system.

The change affects some details in the protocol and in the cryptographic operations, but there are no essential difficulties. The agreement and broadcast protocols need to be changed as follows:

- Where a set of $n - t$ values is required, take all values in $\mathcal{P} \setminus S$ for some $S \in \mathcal{A}^*$.

- Where $2t+1$ values are needed, take all values in $S \cup T \cup \{i\}$ for any $S, T \in \mathcal{A}^*$ with $S \cap T = \emptyset$ and $i \notin S \cup T$.
- Where $t+1$ values are needed, take all values in $S \cup \{i\}$ for any $S \in \mathcal{A}^*$ and $i \notin S$.

1.2 Synchrony Model

The relevance of the choice of synchrony model has its roots in the tradeoff between safety of the fully asynchronous model, and effectiveness of the fully synchronous one. We will guide the reader through the reasoning that led us to adopt a partially synchronous model to support our algorithms and protocols. We will see that the construct we propose (the Trusted Timely Computing Base) can be used to support the correct execution of both timed and time-free protocols.

1.2.1 Tradeoffs Between Asynchrony and Synchrony

Research in distributed systems algorithms has traditionally been based on one of two canonical models: fully asynchronous and fully synchronous models. For a detailed discussion on this see [104]. Asynchronous models do not allow timeliness specifications. They are time-free, that is, they are characterized by an absolute independence of time.

Distributed systems based on such models typically have the following characteristics[1]:

Pa 1 Unbounded or unknown processing delays

Pa 2 Unbounded or unknown message delivery delays

Pa 3 Unbounded or unknown rate of drift of local clocks

Pa 4 Unbounded or unknown difference of local clocks

Asynchronous models obviously resist timing attacks, i.e. attacks on the timing assumptions of the model (i.e., the existence of an upper bound for an action), which are non-existent in this case. However, because of their time-free nature, asynchronous models cannot solve timed problems. However, timeliness is part of the required functionality of

[1] Pa3 and Pa4 are tautologies listed for a better comparison with the synchronous model characteristics listed later. A local clock in a time-free system is nothing else than a sequence counter, and for that reason clock synchronization is also impossible.

interactive applications, such as on-line operations on the stock market, multimedia, air traffic control. For example, they cannot address Quality of Service (QoS) specifications, which are of increasing importance in the measure of the quality of transactional systems in open networks such as the Internet (e.g., stock exchange, e-commerce). "False" asynchronous algorithms have been deployed over the years, exhibiting subtle but real failures, thanks to the inappropriate use of timeouts in a supposedly time-free model.

In addition, asynchronous models preclude the deterministic solution of interesting problems, such as consensus or Byzantine agreement [41]: only probabilistic solutions work in this case. Besides, the only way asynchronous models can reason in terms of causality between events is in a logical way, and this is insufficient if hidden communication channels exist between participants. Causality or cause-effect order is a crucial factor of correctness in some interactive and competitive applications (e.g., trading orders to a stock trading floor, bidding in a virtual market place).

In practice, many of the emerging applications we see today, particularly on the Internet, have interactivity or mission-criticality requirements. That is, service must be provided on time, either because of user-dictated quality-of-service requirements (e.g., network transaction servers, multimedia rendering, synchronized groupware), or because of dependability constraints (e.g., command-and-control applications such as air traffic control or emergency networks).

Synchronous models allow timeliness specifications. In this type of model, it is possible to solve all the hard problems (e.g., consensus, atomic broadcast, clock synchronization) [25]. Synchronous models are characterized by having known bounds for processing and message delivery delays, and for the rate of drift and difference among local clocks. In consequence, such models solve timed problem specifications, one precondition for at least a subset of the applications targeted in MAFTIA, for the reasons explained above. Imagine for example the hardness of implementing real-time stock exchange transactions on the Internet, based on real-time quotes, and with temporal order between competitive requests, to ensure market fairness. Synchronous systems are characterized by the following properties:

Ps 1 There exists a known bound for processing delays

Ps 2 There exists a known bound for message delivery delays

Ps 3 There exists a known bound for the rate of drift of local clocks

Ps 4 There exists a known bound for the difference among local clocks

It is easy to see that synchronous models are susceptible to time-based attacks, since they make strong assumptions about things happening on time. Technically, timeliness

is expressed in two ways: positioning events in the timeline and determining execution durations. Synchronous models are fragile in terms of the coverage of such timeliness specifications. Timing bounds are assumed to be valid for correct processes, and conversely their validity must not be compromised by incorrect processes. Both are difficult to achieve. For example, algorithms based on messages arriving by a certain time may fail if the communication system has performance unstability. Likewise, reading the actual global time from a clock, may fail in dangerous ways if manipulated by an adversary [44]. Likewise, causal delivery order of messages or event trace analysis based on physical timestamps may be disturbed to the advantage of a hacker who, for example, manipulates the time-stamping facility.

1.2.2 Partial Synchrony

The introductory words above explain why synchronism is more than a mere circumstantial attribute in distributed systems subjected to malicious faults: *absence of time is detrimental to quality of service; presence of time increases vulnerability.* Restrictions to the asynchrony of time-free systems have been addressed in earlier studies [35, 38] but timed partially synchronous models have deservedly received great attention in recent times. They yield better results, essentially for three reasons: (i) they allow timeliness specifications; (ii) they admit failure of those specifications; (iii) they provide timing failure detection.

Previous timed partially synchronous models, such as the quasi-synchronous [99] and the timed-asynchronous models [29], share the same observation: *synchronism or asynchronism are not homogeneous properties of systems.* That is, they vary with time, and they vary with the part of the system being considered. However, each model has treated these asymmetries in its own way: some relied on the evolution of synchronism with time, others with space or with both. Synchronism assumptions were not totally transparent to applications, and architectural constructs were rarely used to enforce these assumptions.

We are particularly interested in a model based on the existence of a timely computing base, which is both a timely execution assistant and a timing failure detection oracle that ensures time-domain correctness of applications in environments of uncertain synchronism [101]. The timely computing base model addresses a broader spectrum of problems than those solved by previous timed partially synchronous models. In the timely computing base model, it is assumed that systems have an architecture such that applications, however asynchronous they may be and whatever their scale, can rely on services provided by a special module which is timely and reliable. That is, the time-related aspects may be confined to the timely computing base part, and in consequence, the application may even

be time-free.

This research has focused on benign (non-arbitrary, non-malicious) failure modes. However, the architectural characteristics of the timely computing base enable its extension in order to be resilient to value- as well as time-domain failures. This contributes to supporting what we described earlier as the hybrid fault model of MAFTIA, encompassing malicious faults whilst guaranteeing system timeliness in a much more robust way than fully synchronous models would.

Besides, since the model allows any degree of asynchrony, both timed and time-free applications can be supported by the same infrastructure. This assumes great relevance, since it will be necessary to have algorithms of both classes, in order to support the expected tradeoffs between timeliness and robustness vis--vis malicious faults.

We address these issues with more detail in the sections to follow.

1.2.3 The Trusted Timely Computing Base

We call such an extended model, whose development we pursue in the MAFTIA project, a Trusted Timely Computing Base, or TTCB. In one sense, a TTCB has similar design principles to the very well known paradigm in security of a Trusted Computing Base (TCB) [1]. However, the objectives are drastically different. A TCB aims at fault prevention, in order to ensure that the whole state and resources of services runnning on the TCB are tamper-proof. It is based on logical correctness and makes no attempt to reason in terms of time. In contrast, a TTCB aims at fault tolerance: application components can be tampered with, some may fail, but the whole application should not fail. To ensure that, it can only count on being able to make a few priviledged calls to the TTCB, which ensures a set of minimal trusted and timely services. In other words, a TTCB is an architectural artefact supporting the construction and trusted execution of fault-tolerant protocols and applications.

In essence, the TTCB must follow a few construction principles to guarantee its behaviour in the presence of faults:

- Interposition: it must by construction be interposed between vital resources and any attempt to interact with them.
- Shielding: the TTCB construction is such that it itself is protected (1) from faults affecting timeliness and (2) from faults affecting security.
- Validation: the TTCB functionality is such that it allows the implementation of verifiable mechanisms w.r.t. both to (1) timeliness and (2) security.

The architecture of a system with a TTCB is suggested by Figure 1.2.

The first relevant aspect is that the heterogeneity of system properties is incorporated into the system architecture. There is a generic or *payload system*, over a payload network. This prefigures what is normally "the system" in homogeneous architectures, that is, the place where participants resident in the several hosts run distributed applications.

Additionally, there is a *control part*, made of local TTCB modules, interconnected by some form of medium, the control network. We will refer to this set-up as the distributed TTCB, or simply TTCB when there is no ambiguity. The protocols and software modules implementing the payload applications can rely on the assistance of the TTCB services.

The model puts no limits on how asynchronous the payload applications can be, and how hardly they can be attacked. Concrete systems will be specified (and built) with the degree of synchronism and level of threat which is commensurate with the application in mind.

The second relevant aspect of the TTCB is that its well-defined properties should be preserved by construction, regardless of the properties of applications running with its assistance: it is synchronous, and it is trusted to execute as specified, being resilient to intrusions. This casts the notion of component with fail-controlled (or fault-preventive) behaviour in the architecture, even with regard to malicious faults, as proposed in the hybrid fault model introduced in Section 1.1.

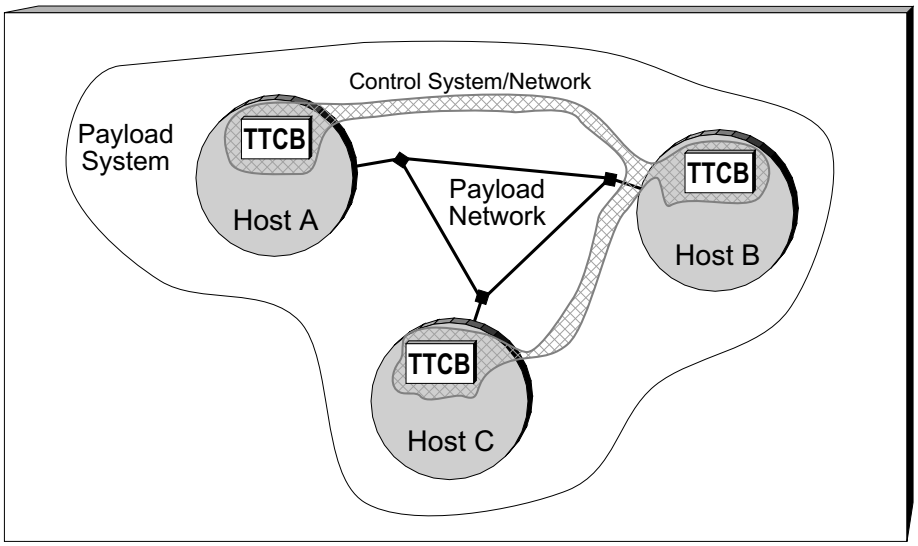

Figure 1.2: Trusted Timely Computing Base Model

Unlike the classic TCB, the TTCB can be a fairly modest and simple component of the system, used as an assistant for parts of the execution of the payload protocols and applications. When assisting timed fault-tolerant protocols, it can help achieve both time-

liness and safety properties of the former (especially those related with security properties). When assisting time-free protocols, it can help achieve safety properties, not only those related with security properties, but those general safety properties that require a minimal synchronism even in time-free systems, such as perfect crash failure detection [25].

Moreover, depending on the type of application, it is not necessary that all sites have a local TTCB. Consider the development of a fault-tolerant TTP (Trusted Third Party) based on a group of replicas that collectively ensure the correct behaviour of the TTP service vis--vis malicious faults. One possibility is for the replica group activity to be based on algorithms that support an arbitrary failure mode of all system components at all instants (e.g., asynchronous randomized Byzantine Agreement), with the corresponding penalty in performance and lack of timeliness. Alternatively, the replica group management may rely on simpler algorithms, generically supporting an arbitrary level of threat, except for the bits executed on the TTCB, which for the matter is a synchronous subsystem with a benign (intrusion-free) failure mode. Running those bits of the algorithm on the TTCB substantiates the otherwise unsustainable claim of "partially benign behaviour".

A TTCB should be built in a way that secures both the synchronism properties mentioned earlier, and its correct behaviour vis--vis malicious faults. In consequence, a local TTCB can be either a special hardware module, such as a tamperproof device, an auxiliary firmware-based microcomputer board, or a software-based real-time and security kernel running on a plain desktop machine such as a PC or workstation. Likewise, a distributed TTCB assumes the existence of a timely inter-module communication channel. This channel can assume several forms exhibiting different levels of timeliness and resilience. It may or not be based on a physically different network from the one supporting the payload channel. Virtual channels with predictable timing characteristics coexisting with essentially asynchronous channels are feasible in some of the current networks. The IETF is studying several approaches to achieve this behaviour in the Internet [65, 49]. Virtual channels with predictable timing characteristics coexisting with essentially asynchronous channels are feasible in some current networks, even over the Internet [87]. Such virtual channels can be made secure through virtual private network (VPN) techniques, which consist of building secure cryptographic IP tunnels linking all TTCB modules together, and these techniques are now supported by standards [52]. On a timeliness side, it should be observed that the bandwidth required of the control channel is bound to be much smaller than that of the payload channel. In more demanding scenarios, one may resort to alternative networks (real-time LAN, ISDN connection, GSM Short Message Service, GPRS (General Packet Radio Service) or UMTS, Low Earth Orbit (LEO) satellite communication).

The Basic TTCB Services

The TTCB provides the following services related to time: trusted timely execution; trusted absolute timestamping; trusted duration measurement; trusted timing failure detection. They are trusted versions of the timely computing base services presented in [102], except for absolute timestamping, which was included recognising the need for interactive applications on open systems to be referenced to absolute time (e.g., TAI or UTC). The outputs of all these services are trusted to be correct and authentic (e.g., they can be signed by the TTCB with its private key, see below).

The services have a distributed scope, although they are provided to processes via the local TTCB instantiations. Any service may be provided to more than one user in the system. For example, recall that the MAFTIA Reference Model foresees the capability of providing "insecurity signals" both to layers above and to System Administration. The failure notifications produced by the trusted timing failure service are an example of such a signal, which may be given to all interested users.

We define below the properties of the services. The properties are defined as seen at the TTCB interface. We start with trusted timely execution. *Trusted Timely Execution* relies on the following property of the TTCB:

TTCB 1 *Given any function F with an execution time bounded by a known constant T_{Xmax}, and given a delay time lower-bounded by a known constant $T_{Xmin} \geq 0$, for any execution of the function triggered at real time t_{start}, the TTCB does not start the execution of F within T_{Xmin} from t_{start}, and terminates F within T_{Xmax} from t_{start}*

Timely Execution allows the TTCB to execute arbitrary functions deterministically, given a feasible T_{Xmax}. The executions can be delayed (T_{Xmin}), such as those resulting from timeouts.

Absolute Timestamping is achieved by the following:

TTCB 2 *Given any event e_i occurring in any node, at real time t_i, the TTCB measures t_i as T_i, and the error of T_i is bounded by a known value.*

The trusted absolute timestamping service provides a timestamp (it can be Authenticated, e.g., signed with a TTCB private key). A timestamp produced in any local TTCB is immediately meaningful to any other TTCB (in the sites where one exists) since local clocks are synchronized, and it is also meaningful for any external entity referenced to real time, since clocks are also externally synchronised to an absolute time reference. This is

stronger than the initial model in [101], but very much helpful to order events inside the system vis--vis external but absolute-time-referenced events. Some applications in mind for MAFTIA, e.g. financial and electronic commerce, will have such needs.

Duration Measurement relies on another property:

TTCB 3 *Given any two events e_s and e_e occurring in any two nodes, respectively at real times t_s and t_e, $t_s < t_e$, the TTCB measures the duration between e_s and e_e as T_{se}, and the error of T_{se} is bounded by a known value.*

The property TTCB 3 may seem redundant with TTCB 2. But in fact, it may happen that there is an internal ordering mechanism that supplies TTCB 3 with better precision than the one achieved by using absolute timestamp subtraction given by TTCB2.

Another crucial service of the TTCB is trusted timing failure detection. If we define *timed actions* as operations that have timeliness properties, then they have to be executed in a given interval of time after an instant. A timing failure happens when a timeliness property of a timed action is violated. The Timing Failure Detection service is defined using an adaptation of the terminology of Chandra [25]:

TTCB 4 Timed Strong Completeness: *There exists T_{TFDmax} such that given a timing failure at p in any timed action $X(p, e, T_A, t_A)$, the TTCB detects it within T_{TFDmax} from t_e*

TTCB 5 Timed Strong Accuracy: *There exists TTFDmin such that any timed action $X(p, e, T_A, t_A)$ that does not terminate within $-T_{TFDmin}$ from t_e is considered timely by the TTCB if the local TTCB does not crash until $t_e + T_{TFDmax}$*

The majority of detectors known are *crash* failure detectors. We introduce timing failure detectors (TFD). Timed Strong Completeness can be understood as follows: "strong" specifies that any timing failure is perceived by all correct processes; "timed" specifies that the failure is perceived at most within T_{TFDmax} of its occurrence. In essence, it specifies the detection latency of the TFD. Timed Strong Accuracy can also be understood under the same perspective: "strong" means that no timely action is wrongly detected as a timing failure; but "timed" qualifies what is meant by "timely", by requiring the action to occur not later than a set-up interval T_{TFDmin} before the detection threshold t_e. In essence, it specifies the detection accuracy of the TFD.

The TTCB Security-Related Services

The TTCB was initially defined as an extension of the timely computing base for malicious environments. However, it was found opportune to include some security-related services inside the TTCB since these services can take advantage of the properties of the TTCB, i.e., of its trustworthiness and timeliness by construction. Three criteria were used to define the security related services of the TTCB:

- They must be highly trusted.
- They must be "lightweight" for the TTCB to be verifiable.
- They should be a minimal set that assists and not replaces the implementation of middleware building blocks.

An obvious use for the local TTCBs is authentication of the site where it belongs. The TTCB is trusted, tamper-proof, so it is probably one of the few places in a system secure enough to save a long-term private key. The corresponding public key can be published (e.g., using a PKI) so that other entities can authenticate whatever is produced by the TTCB. Incidentally, the private key can also be used to sign the output of all TTCB services, as suggested above, in case there is a need for checking integrity and authenticity.

The security-related services of the TTCB are defined informally below:

Trusted Random Number Generation: *This service generates trustworthy random numbers.*

Trusted Block Consensus: *This service achieves consensus on a fixed size block of data between local TTCBs.*

Participant-TTCB Authentication (and key establishment): *This service mutually authenticates a local participant (e.g., a local process or application, or an applet inside a* Smartcard*) and the local TTCB, and establishes a shared key between them.*

Participant-Participant Authentication (and key establishment): *This service can be used for two or more participants resident in the same or different hosts to authenticate themselves mutually through the TTCB. A shared key is established.*

Random numbers are essential for building cryptographic primitives and it is important that they are trusted to be random. Block consensus is not intended for consensus

operations on payload data, but it may be used to perform very simple but crucial decision steps in more complex payload protocols. The shared key established in the mutual authentication, either between a participant and its local TTCB, or between two participants, can be used for them to communicate securely, establishing a trusted path. Other TTCB services can optionally be invoked over this path.

1.2.4 Timed Approach

The timely computing base work is based on the assumption that a general model can be devised that encompasses the entire spectrum of partial synchrony [103, 100, 30], from fully asynchronous to fully synchronous.

Of course, it serves extremely well those timed applications that fall in between, and which, in spite of having a notion of timeliness (i.e., time bounds, deadlines, etc.), cannot always fulfill these requirements adequately. The common denominator of systems belonging to that realm is that they can exhibit *timing failures*, i.e., they can violate timeliness properties.

In consequence, a programming model can be devised, based on three notions:

- The payload system, although timed, has uncertain timeliness.
- The control system (the TTCB), can assist an application running on the payload system, to determine useful facts about time (be sure it executed something on time; measure a duration; determine it was late doing something).
- The payload system, despite being imperfect (it suffers timing faults, some of which may result from attacks) can react (implement fault tolerance mechanisms) based on reliable information about presence or absence of errors (provided by the TTCB at its interface).

That is, given the baseline partial synchronism of a payload application, there are no guarantees about the actual time of invocation or completion of services and functions by the former, including TTCB services. Furthermore, those instants and durations can be manipulated by an adversary, with the intent of intruding the system. Accidental or not, timing failures in system components may lead to the violation of safety properties of the system, and lead to its failure, if nothing is done. However, the TTCB can provide crucial information about what is happening in the payload system, so that it can recover from abnormal situations and still achieve the provision of correct service.

In consequence, what we propose here is a programming model that lets the programmer build applications satisfying timeliness requirements, because these are crucial for the emerging interactive services on the Internet. Those applications must work under assumptions which can be considered optimistic with regard to time, since: (i) they will be technically hard to achieve in such large-scale environments; (ii) they will be susceptible to attacks on the very timing aspects. Should the system be left to itself, this might have undesirable or even catastrophic consequences, given that those assumptions might fail unwittingly to the system (lack of coverage). However, we provide the system with an oracle (the TTCB) capable of supplying information on how the (payload) system is doing w.r.t. timeliness, so that the system may itself react and avoid failure, should it have accidental or malicious timing faults.

Additionally, the TTCB, which is the only part of the system required to be tamper-proof, also has the capability of providing a few basic but crucial security-related services, which are trusted as supplied at the TTCB interface with the payload system components. We expect these services— such as random number generation, basic consensus and authentication— to be of use for the implementation of efficient fault tolerant protocols resilient to arbitrary (accidental and malicious) faults.

The definition of the interface to these services is thus a very important problem, which will be addressed in a forthcoming document. Essentially, the interface has to make the bridge between a synchronous and trusted environment (the TTCB), and a potentially asynchronous and untrusted one (the payload system).

1.2.5 Time-free Approach

One may also adopt the asynchronous model and work with in a completely time-free environment. Of course, asynchronous protocols cannot guarantee a bound on the overall response time of an application, but they were never meant to. In general, an asynchronous model provides a conceptually simple and nice framework for developing and reasoning about the correctness of an algorithm, such that it satisfies liveness and integrity under *all* timing conditions. This has some advantages for the design of secure distributed systems, which is one reason for pursuing such a model in the context of MAFTIA. In fact, sometimes it is necessary and worthwhile to sacrifice timeliness for resilience.

Let us analyse a little better how timed algorithms can be attacked. Specifying timeout values may be very difficult when protecting against arbitrary failures that may be caused by a malicious attacker. It is usually much easier for an intruder to block communication with a server than to subvert it. As mentioned before, prudent security engineering assumes that an attacker has full access to all specifications, including timeouts,

and keeps only cryptographic keys secret. Even with only partial control of the network, such an adversary may simply delay the communication to a server for a little longer than the timeout and the server appears faulty to the rest of the system.

Time-based failure detectors [25] can easily be fooled into making an unlimited number of wrong failure suspicions about honest parties like this. This problem arises because one crucial assumption underlying the failure detector approach, namely that the system is stable for some longer periods when the failure detector is accurate, may fail against a malicious adversary. A clever adversary may subvert a server and make it appear to be working properly until the moment at which it deviates from the protocol—but then it may be too late. Heuristic predictions about the future behavior of a server are pointless in security engineering. In consequence, the failure detector abstraction is not as useful under a Byzantine (arbitrary failure) context as it is under a crash-failure context. This opens two perspectives: either the failure detector is made to work properly in a malicious fault environment, or a solution is devised that does not require failure detectors.

The first solution is nothing else but following a programming model similar to the one proposed for the timed approach. However, in this case, the payload system is fully asynchronous, that is, time-free, and the time-based failure detector, in the TTCB, is exclusively used to generate and control timeouts in a trusted way. Trusted should be read both from the viewpoint that it is impossible to have accurate timeout-based failure detectors in fully asynchronous systems [25], and from the viewpoint that with whatever synchrony (even in partially synchronous systems) the former may be manipulated by a malicious attacker, unless implemented in a tamper-proof environment.

For the second solution, we will rely on randomized (probabilistic) protocols, e.g. Byzantine agreement. These protocols make essentially very few assumptions about the environment (and in consequence, they also provide little guarantees, e.g. w.r.t. completion). Note however that the TTCB may still be used to strengthen some of the assumptions underlying this approach, in quest for more efficient implementations. For example, for making the algorithms aware of the current synchrony of the system and temporarily change operation mode, or for supplying basic security-related functions in a trusted way.

Despite the practical appeal of the asynchronous model, not much research has concentrated on developing efficient asynchronous protocols or implementing practical systems that need consensus or Byzantine agreement [45]. Often, developers of practical systems have avoided the approach because of the result of Fischer, Lynch, and Paterson [41], which shows that consensus is not reachable by deterministic protocols, even with crash failures only. But but there are randomized solutions that use only a constant expected number of rounds to reach agreement [77, 12, 23]. Moreover, by employing modern, efficient cryptographic techniques, this approach has recently been extended to a practical yet provably secure protocol for Byzantine agreement in the cryptographic model that withstands the

maximal possible corruption [18]. The fact that randomized agreement protocols may not terminate with non-zero probability does not have to bother us because this probability is negligible. Moreover, if a protocol uses authentication, digital signatures, or *any* cryptography at all, and the practical protocols mentioned above do so, a negligible probability of failure cannot be ruled out. Note that this failure probability is determined by the key lengths, and therefore purely a function of the system design.

A variation of the asynchronous model is to assume probabilistic behavior of the communication links [15, 68], where the probability that a link is broken permanently decreases *over time*. This probability is a function of the system behavior. But since this involves a timing assumption, it is essentially a probabilistic synchronous model (perhaps it should also bear that name) and suffers from some the problems mentioned before. Another related model [68] assumes a fairness property and a partial order imposed by the underlying communication system, but such assumptions seem also difficult to justify on the Internet.

1.3 Topological Model

Previous work on large-scale open distributed systems has shown the value of topology awareness in the construction of efficient protocols, from both functionality and performance viewpoints [84]. The principle is explained very simply: (i) the topology of the system is set up in ways that may enhance its properties; (ii) protocols and mechanisms in general are designed in order to recognize system topology and take advantage from it.

This is achieved both by creating adequate physical topologies (the part of the system under the control of the organization, e.g., the local networks) and by logically reorganizing existing physical topologies (the part of the system outside the control of the organization, e.g. the Internet).

We intend to extrapolate the virtues of topology awareness to security in the MAFTIA architecture, through a few principles for introducing topological constructs that facilitate the combined implementation of malicious fault tolerance and fault prevention. The first principle is to use topology to facilitate separation of concerns: the site-participant separation in the internal structure of system hosts (or nodes— in the context of this document, we will use these two designations interchangeably) separates communication from processing; and the WAN-of-LANs duality at network level separates communication amongst local aggregates of sites, which we call *facilities*, from long-haul communication amongst facilities. The second principle is to use topology to construct clustering in a natural way. Two points of clustering seem natural in the MAFTIA large-scale architecture: sites and facilities. These principles are illustrated in Figure 1.4.

1.3.1 Sites and Participants

The MAFTIA architecture supports interactions among entities (e.g. processes, tasks, etc.) in different hosts. We call them generically *participants*. Participants, which execute distributed activities, can be senders or recipients of information, or both, in the course of the aforementioned activities. The local topology of hosts is such that they are divided into a site part, which connects to the network and takes care of all inter-host operations, i.e., communication, and a participant part, which takes care of all distributed activities and relies on the services provided by the site-part modules. Participants interact via their respective site part, which handles all communication aspects on behalf of the former, as represented in Figure 1.3.

From now on, we will refer to sites, when taking the communication/networking viewpoint on the system, and we will refer to participants, when taking the activity/processing viewpoint.

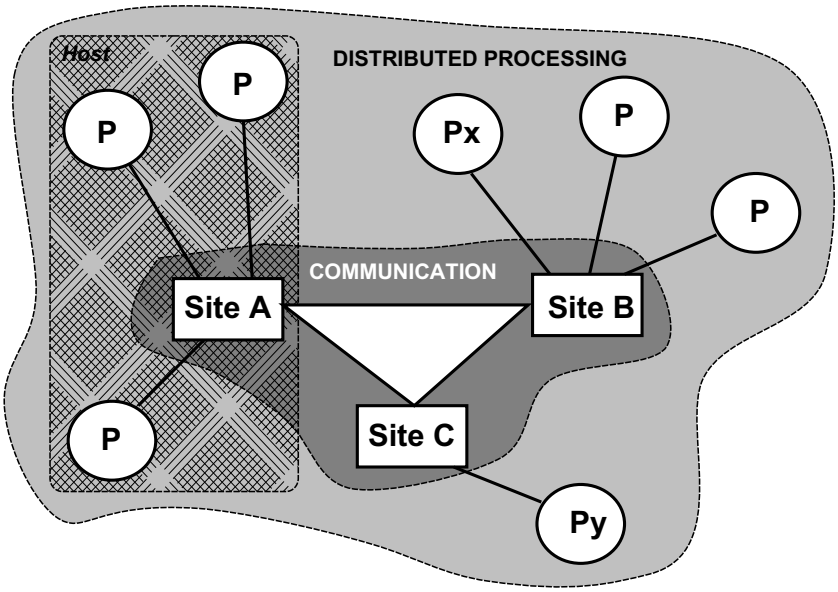

Figure 1.3: Site-participant duality

A system built according to the site-participant duality model provides a framework for defining realms of different synchrony, reliability and security. For example, intra-site communication can be assumed to have better properties of synchrony and reliability with regard to both accidental and malicious faults, in contrast with inter-site communication. In consequence, while site failure detection is unreliable in a network of uncertain synchronism, participant failures can be reliably detected. Likewise, different assumptions can be made concerning trustworthiness of the participant and site parts.

This distinction between *sites* and *participants* is in favor of – and can only be achieved by – a *communication subsystem* approach for structuring the machine's networking. A site-level *protocol server* should take care of all send and receive activities on behalf of the participants residing locally to it.

1.3.2 Two-tier WAN-of-LANS

The global topology of the networking infrastructure is seen as a logical two-tier WAN-of-LANs. Large-scale computing infrastructures exhibit what appears to be a two-tier organization: pools of sites with privately managed high connectivity links, such as LANs or MANs or ATM fabrics, interconnected in the upper tier by a publicly managed point-to-point global network (e.g., the Internet). More concretely, we mean that the global network runs standard, de jure or de facto, protocols whilst each local network is run by a single, private, entity, and can thus run specific protocols alongside standard protocols.

Again, this structure, depicted in Figure 1.4, offers opportunities for making different assumptions regarding the types and levels of threat and degrees of vulnerability of the local network versus the global network part. Incidentally, this does not necessarily mean considering intra-facility networking threat-free. For example, certain port scans or pings in the global network may mean absolutely nothing, whereas they may mean an attack if performed inside the facility. On the other hand, an intruder working from the inside of the facility may have considerably more power than one working from the outside. Note that the first hypothesis is in the direction of considering the facility as a more benign environment, wheras the second is not.

1.3.3 Clustering

Clustering seems one of the most promising techniques to cope with large-scale distributed systems, providing the means to implement effective divide-and-conquer strategies. Whilst this enhances scalability and performance, it also provides hooks for the combined implementation of fault-tolerant mechanisms (e.g., cryptographic group management protocols) and fault-preventive protection strategies (e.g., firewalls). We identify at least two clustering opportunities: (i) the Facility as a cluster of hosts, or more appropriately sites, if seen from the viewpoint of the network; (ii) the Site as a cluster of participants, the ones that reside in the relevant host, if we take the network viewpoint once again.

The first clustering level is obviously compatible with the two-tier architecture identified in the previous section, and is illustrated in Figure 1.4. Clustering sites that coexist

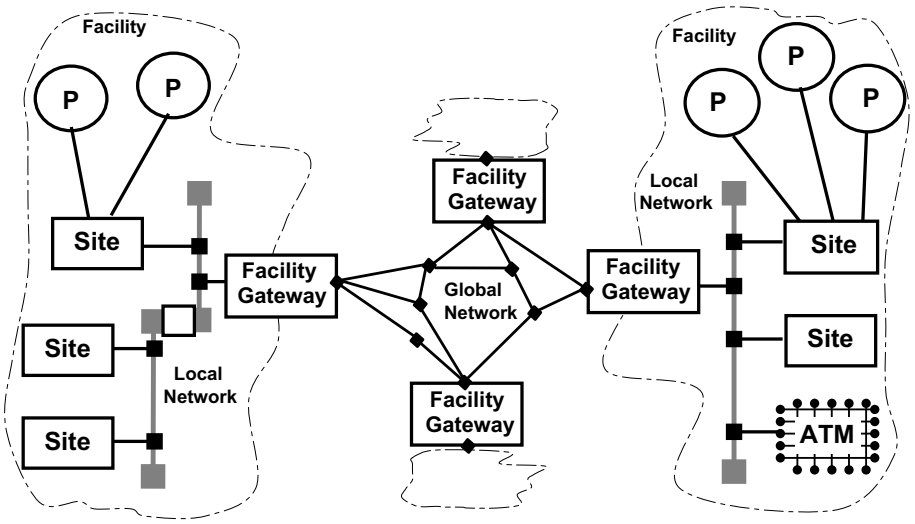

Figure 1.4: Two-tier WAN-of-LANs

in the same local network can simplify inter-network addressing, communication and administration of these sites. Sites are hidden behind a single entry-point, a *Facility Gateway*, a logical gateway that represents the local network members, for the global network.

The second level of clustering consists of taking advantage of a multiplying factor between the number of sites and the (sometimes much larger) number of participants that are actively engaged in distributed applications. Organization-dependent clustering allows specific protocols to be run behind the Facility Gateways, without conflicting with the need to use standard protocols in wide-area networking, i.e. beyond those gateways. Global network communication is then performed essentially among Facility Gateways.

From a security viewpoint, participant-site clustering allows investing in the implementation of fault-tolerant and fault-preventive mechanisms at site level, to collectively serve the participant-level applications residing in the host. On the other hand, the opportunities offered by site-facility clustering with regard to security are manifold: firewalling at the Facility Gateway; establishing inter-facility secure tunnels ending in the facility agents; inspecting incoming and outgoing traffic for attack and intrusion detection; ingress and egress traffic filtering; internal topology hiding through network address translation, etc.

1.3.4 Recursivity

The topological construct we have just presented can be recursively applied at different levels, in order to represent very-large-scale organizations. On an intra-facility level, further hierarchies, namely those already deriving from hierarchical organization of

subnetworks and domains, are not precluded, if the Facility Gateway role is respected. Protocols can easily take advantage of further topology refinements.

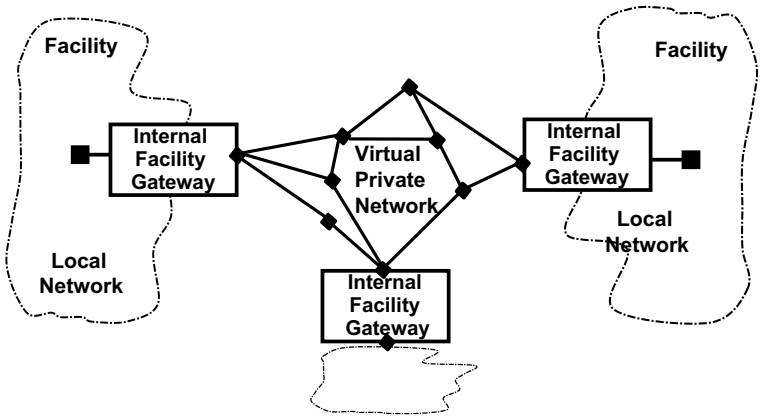

Figure 1.5: Virtual Private Networks in MAFTIA

At an intra-organization level, the topology depicted in Figure 1.4 can be instantiated to represent an organization with multiple geographically dispersed facilities interconnected by secure tunnels whose end points are Facility Gateways. The only role of the Facility Gateways in this special configuration is to implement the Virtual Private Network (VPN) ensuring this internal communication, so they would be better named as Internal Facility Gateways, as illustrated in Figure 1.5.

Finally, the organization still needs to be connected to the Internet, and contact with other organizations. So, the topology in Figure 1.4 can represent the big picture— inter-organization interactions— where each facility is so to speak the top-level facility, the one containing the organization portal to the Internet. Communication with other organizations through the Internet is ensured through normal gateways, let us call them External Facility Gateways.

On an extra-organizational level, the Facility Gateway, which is a logical entity and does not necessarily correspond to a single machine, offers the necessary services, such as incoming email, web presence, e-commerce, and so forth, implemented by extranet-based servers, serving potentially many thousands of clients.

1.4 Interaction Styles

In MAFTIA we intend to support different styles of basic interactions among the participants. These interaction styles also assume topological importance, because they can be combined to construct complex software architectures, but the possible combinations are conditioned by the system topology.

Although the main goal of MAFTIA is to provide security in the face of malicious faults, the MAFTIA architecture must also provide a versatile functional support in order to be useful. Consequentially, MAFTIA will support the main interaction styles used in distributed computing, namely:

- client-server, for service invocations
- multipeer, for interactions amongst peers
- dissemination, of information in push or pull form
- transactions, for encapsulation of multiple actions

1.4.1 Client/Server

Client-server interactions can be implemented by two different mechanisms: in *closed loop*, that is, in a blocking, request-confirmation manner, usually performed through RPC [14]; or in *open loop*, that is, in an unblocking manner, usually performed through group communication, pioneered by Cheriton [27], Birman [16] and Cristian [28], and followed by many others. From previous works [17, 55], it is known to be difficult to scale service access when clients need to be strongly coupled (for instance, when all messages need to be ordered). However, there are a number of services, especially in large-scale systems, where the coupling among clients, and between a client and the server, can be weakened. In addition, techniques supporting reliable large-scale remote server access have been deployed, allowing services to be easily replicated and invoked transparently, without necessarily implying a degradation of strong failure semantics [82]. Both approaches are easily implemented using group-based open-loop mechanisms.

1.4.2 Multipeer

Another style of interaction is multipeer, conveying the notion of spontaneous, symmetric interchange of information, amongst a collection of peer entities. This paradigm appeared as early as in [75] where it is called multipoint association, and also in [71] where it is called conversation, a term that we avoid in order not to cause confusion with a different paradigm with the same name described in [19], and also discussed below. Multipeer interactions are the kind of interaction one might wish among managers of a distributed database, a group of commerce servers, a group of TTP servers, or a group of participants running a cryptographic agreement (e.g., contract signing). Communication

requirements may be heavy in ordering and reliability requirements, and a notion of composition or membership may be required (for example, to provide explicit control over who is currently in the group). Again, the highly interactive nature of the multipeer style of interactions prevents per se the number of participants in real applications from exceeding the small-scale threshold.

1.4.3 Dissemination

Dissemination is a style of interaction which combines the information push and pull approaches. Information is published by publishers, and is made available on a repository. Message subscription can be implemented using two different alternatives: the push strategy or the pull strategy. With the push strategy, subscribers just register their interest in receiving a certain class of messages with the server, and the server is then responsible for dissemination of these messages to the interested subscribers. With the pull strategy, it is up to the subscriber to contact the server periodically to fetch the messages on request. The number of recipients may be rather high, while the number of senders will not.

1.4.4 Transactions

Transactions are a style of interaction that allows the encapsulation of multiple actions. Transactions provide fault confinement and simplify reasoning about the properties of applications. Within the MAFTIA framework we will support two types of transactions: atomic transactions, and coordinated atomic actions.

The well-known atomic transactions [61] style of interaction allows the grouping of a set of operations into a single higher level action that either completes successfully or has its effects undone. Additionally transactions guarantee the properties of atomicity, consistency, isolation and durability (ACID). Atomicity is the property that the transaction either completes and its effects on resources are made permanent (committed) or the effects are undone (aborted). This is achieved either through backward recovery where no application-specific knowledge is needed or forward recovery where an application-specific protocol is used to leave the resources in a consistent state. Transactions ensure that all changes to the state of resources are consistent irrespective of concurrent access and failures. Each transaction is isolated from another, they cannot access shared resources at the same time. Finally, the durability property means that results of a committed transaction will survive the crash of the nodes involved in the transaction. This requires that all committed state changes are written to some crash-proof storage.

The Coordinated atomic actions (or CA actions) style of interaction [106][107] extends transactions and is a mechanism for coordinating multi-threaded interactions and ensuring consistent access to objects in the presence of concurrency and potential faults. It can be regarded as providing a framework for nested multi-participant interactions that in addition provides very general exception handling facilities. CA actions will allow the building of advanced applications that rely upon multi-participant interactions. Also, CA actions potentially could be used as a structuring mechanism for error containment since an action provides a structural boundary that keeps information and events under control. The CA action mechanism's exception handling facilities allow both application-level faults and reported unmasked low-level faults to be tolerated. These features makes CA actions a good candidate for a structuring mechanism for building intrusion-tolerant components.

1.5 Group Models

The concept of group appears intuitively when describing many distributed actions[97]. The most striking examples can be found in the Computer Supported Collaborative Work (CSCW) arena, where applications are supposed to support the interaction among *groups* of end-users (e.g., distributed document processing, concurrent engineering), and its more recent instantiations on the Internet (e.g., desktop conferencing, virtual enterprises, marketplaces). Often, groups appear in a less evident but nonetheless useful way, when replication is involved. Examples of relevant problem specifications are: copies of a table in a *group* of sites in a network, for performance reasons; a *group* of replicas, for fault-tolerance reasons.

The group abstraction fits naturally into the MAFTIA architecture system model. Participants that trust each other, cooperating on a given task, should be included in the same high-level group. At the low level, to support secure communication, the sites containing participants could form another group. This low-level group would use cryptography and authentication mechanisms, to prevent adversaries from eavesdropping or corrupting the communication. The use of several groups with different service guarantees, and its composition thereof, in the construction of complex applications, can be explored taking into account the topological structure of the network. For example, we could have a hierarchy of groups, whose top level exists at the global network, with its members representing groups inside the local facilities. Higher levels of the hierarchy would have stronger security requirements (e.g., cryptography keys with more bits). Some of the interaction styles that MAFTIA wants to support are also closely related to the idea of groups, for instance, multippeer and dissemination.

1.5.1 Open vs. Closed Groups

Group communication systems can be divided into two categories depending on who can send messages to the group. Systems that support a *closed* model only allow members of the group to send messages to the group. Outside participants that want to interact with the members of the group, either have to join the group before they start to communicate, or they have to send messages (point-to-point) to each member of the group. On the other hand, systems with the *open* model do not impose any restrictions on the senders. Any participant that knows the identifier of the group can send messages to its members.

From a security point of view, it is much easier to support the closed group model. The group can be considered an island of trust, where all messages are encrypted using, for instance, a group key that is known to all members. Exterior messages can be immediately disregarded, without the necessity of authenticating each message from every sender, because they were not generated using the shared keys. The use of a closed group model imposes, however, some limitations on the construction of the applications. For example, if we want want to build a replicated server, clients (which are usually non-members) must be able to send requests to the group. On this model, there are basically two solutions to this problem: either the client is required to join the group, or one of the servers is selected to receive the requests and then is responsible to send them to the other members. In both cases there is a performance penalty that must be paid. The first solution requires an update on the group membership, and the second the transmission of an extra message.

In an open group, the properties of security (e.g., confidentiality and integrity) can be ensured on the communication among the members using mechanisms similar to the ones used in closed groups. External communication, however, will be more complicated because at least for some applications it will be necessary to authenticate the messages. For example, on a music repository there might be a charge for each MP3 file that is downloaded, which means all clients requests need to be reliably authenticated. The authentication per se, can be done by different components of the architecture, by the group communication system or by the application. Nevertheless, external communication exposes the group to a higher level of threat because there are more types of attacks that can be explored by the adversary, for instance, denial of service attacks.

In the MAFTIA architecture we intend to use an extension of the previous models, which distinguishes between several types of participants [84]:

- *Remote clients* - can *only* send messages to the group
- *Senders* - have to explicitly attach themselves to the group, and they can *exclusively* send messages to the group

- *Members* - can *only* receive messages from the group (and they have necessarily to attach themselves to the group)

This model gives a reasonable amount of flexibility to the participants in their interactions with the group. For instance, only members, that is, the recipients, incur the cost of maintaining the necessary structures and state to ensure the desired message delivery QoS. On the other hand, a participant that wants bidirectional access becomes both a sender and a member. Other participants might only want to send messages occasionally, and in this case they stay as remote clients. The separation in different types of participants is important both in terms of security and performance. Participants that are senders or members have to do a time consuming authentication only once, when they join the group. On the other hand, remote clients have to be authenticated every time they send a message.

1.5.2 Static vs. Dynamic Groups

Groups are classified as *dynamic* or *static* depending on whether it is possible or not to change the composition of the group. Dynamic groups allow updates to the current membership of the group, either by removing the members that are perceived as faulty, or by letting new sites join the group. Static groups keep the same members during the whole lifetime of the system, despite observable corruptions.

It is usually easier to support static groups in a secure manner because cryptographic parameters or keys only have to be distributed once, during the system setup. For example, threshold-cryptography protocols are based on a fixed set of parameters (e.g., n and t) that must be known during initialization, when the key shares are generated. From this moment, these parameters remain constant. While the system is executing, however, an adversary might be able to corrupt one of the sites and learn all the secrets that are kept there. For certain types of applications this might be a serious problem, but for others, for instance applications based on the state-machine replication, no real damage is done as long as the number of faulty sites is smaller than 1/3 of the total number of sites.

For state-machine replication it would be interesting if one could restart periodically the sites with uncorrupted copies of the state and redistribute fresh keys. If this could be done in a secure manner the evil effects introduced by the adversary would be removed. This solution, however, might be difficult to implement unless systems are constructed in a careful way. Normally, an adversary will penetrate a machine not because he or she was able to break the cryptographic algorithms, but because there was a vulnerability that could be exploited. This creates the problem that unless the vulnerability is removed during the refresh, the adversary will be able to regain control of the site. Moreover, if

the same vulnerability is present in other machines, it will be relatively easy and fast to corrupt more than 1/3 of the replicas.

Dynamic groups allow an alternative solution where faulty sites are simply removed from the system. In this model, at any instant of time, there is a *view* that represents the sites that are currently active members of the group. If for some reason the membership has to be changed, then a new view is installed, and new keys are distributed. Dynamically changing the keys, however, is a hard problem if we consider that a corrupted a site might be involved in the view change, because the detection of the intrusion was not done. Nevertheless, this model is much more practical than static groups because real systems are usually long-running, and they grow or shrink as time goes by.

In the MAFTIA architecture we intend to separate the concepts of membership and view. The membership is the set of sites that are members of the group. This set is not static, and can evolve as new sites want to join the group, or others want to leave. The view is always a subset of the membership. It contains all the members that are currently active in the group. If a site fails for some reason, the membership continues to be the same, but the view has to be updated with the removal of the site. This separation between membership and view is important in terms of fault-tolerance systems because it facilitates the group management when there are failures/recoveries or network partitions/rejoins. In terms of security it is still an open problem.

1.5.3 Inclusion and Exclusion of Members

If one adopts dynamic groups, there are protocols needed for including and excluding parties to a group. Unless there are cryptographic secrets shared among a group, this seems not to pose any difficulties. But since the groups in a security context must operate primarily through cryptographic operations, and most practical agreement protocols involve cryptography, care must be taken here.

When a party is excluded from a group because it had been corrupted by an intruder, one must assume that all information known to this party is now also known to the intruder. In the context of threshold cryptography, where each party holds a share of a public key, this party must now be considered corrupted forever. To prevent this undesirable effect, there have been protocols designed for periodically refreshing the key shares held by all parties such that this knowledge becomes useless to the adversary. So-called *proactively secure* cryptosystems [22] can tolerate that all parties are corrupted during the whole lifetime of the system, as long there is always an honest majority between any two resharing steps. They work essentially in synchronous networks with broadcast channels available, and must be adopted for an Internet setting.

When a new party joins a group, it must receive the secret keys pertaining to the group, possibly including shares of threshold cryptosystems. The same issues as for key generation and key management (Section 2.3) arise here as well: either there is a trusted dealer that can distribute the keys (but this is a single point of failure) or a distributed protocol is invoked to produce them (which might be inefficient).

2 Architecture

Paulo Veríssimo, Nuno Ferreira Neves, Miguel Correia, *University of Lisboa (P)*
Christian Cachin, *IBM Zurich Research Lab (CH)*
Brian Randell, Robert Stroud, Ian Welch, *University of Newcastle upon Tyne (UK)*

Consider the generic services commonly found in distributed systems:

- Name Service – supplies the global names and addresses of users, services and resources
- Networking Service – provides access by users and programs to the basic networking and communication facilities (e.g. sockets over TCP/IP on LAN, dial-up, Internet)
- Remote Invocation Service – provides for remote operation client-server invocation
- Time Service – supplies and keeps synchronized a global time reference, normally made of local clocks
- File and Archive Service – provides the abstraction of a unique file system, globally accessible, made of distributed repositories
- Brokerage Service – performs trading and binding of services and users in a heterogeneous environment (e.g., Object Request Broker)
- Registration, Authentication and Authorization Services – Registers users and services, performs runtime authentication of users and control of their access to services and resources
- Administration Service – performs tactical management tasks, in order to manage users and keep the system resources and services operating correctly

Observing these services, we conclude that some of them are standard, and more secure versions of these standards exist today, such as the name service, and the basic protocols of networking, time and remote invocation services in Internet settings. File and archive, and brokerage services have been the subject of a great deal of work focused on improving the security of such subsystems, which can be reused. However, other services are extremely sensitive, and their resilience may make the difference in a secure and fault-tolerant system.

Registration, authentication and authorization services lie at the core of the security of any distributed system. As such, they deserve particular attention in MAFTIA. Namely, specific Authorization and Trusted Third Party services will be developed in the project.

Finally, the administration services are essential to keep the system operating correctly. Many solutions to the general administration problem exist. However, it is felt that research is still worthwhile in security administration, and the project is specifically addressing the problem of developing an Intrusion Detection service.

Finally, as in other areas, it is felt that the development of complex distributed applications should benefit from, if not require, support middleware, in the form of libraries, functions, or protocols, which assist their construction and correct execution. The MAFTIA middleware is oriented to support the main paradigm investigated in MAFTIA, and thus provide a set of those functions and protocols enabling the construction of *intrusion-tolerant* applications and services. These middleware components should be modular, and able to be used recursively to construct more complex components. That should apply both to MAFTIA services like the ones just mentioned, and to end-user applications.

Both these objectives and the discussions of the previous sections have guided the definition of the MAFTIA middleware architecture that we present next.

2.1 System Components

The architecture of a MAFTIA host is represented in Figure 2.1, in which the local topology and the dependence relations between modules are depicted by the orientation of the ("depends-on") arrows. This architecture is orthogonal to the TTCB so that every module can use the services of that component. In Figure 2.1 the set of layers is divided into site and participant levels. The site level has access to and depends on a physical networking infrastructure, not represented for simplicity. The participant level offers support to local participants engaging in distributed computations. The lowest layer is the Multipoint Network module, MN, created over the physical infrastructure. Its main properties are the provision of multipoint addressing and a moderate best-effort error recovery ability, both depending on topology and site liveness information, and basic inter-site secure channels and message envelopes. The MN layer hides the particulars of the underlying network to which a given site is directly attached, and is as thin as the intrinsic properties of the latter allow.

In the site level, the Site Failure Detector module, SF, is in charge of assessing the connectivity and correctness of sites, and the MN module depends on this information. Site failure detection of sites with a local TTCB is reliable because the TTCB is synchronous and secure. For sites without a local TTCB, the SF module depends on the network to perform its job and thus is not completely reliable, due to the uncertain synchrony and susceptibility to attacks of at least parts of the network. The universe of sites being monitored can be parameterized, for example: all sites inside a facility, all sites having to do with ongoing computations at this site, all facility agents, etc. The Site Membership

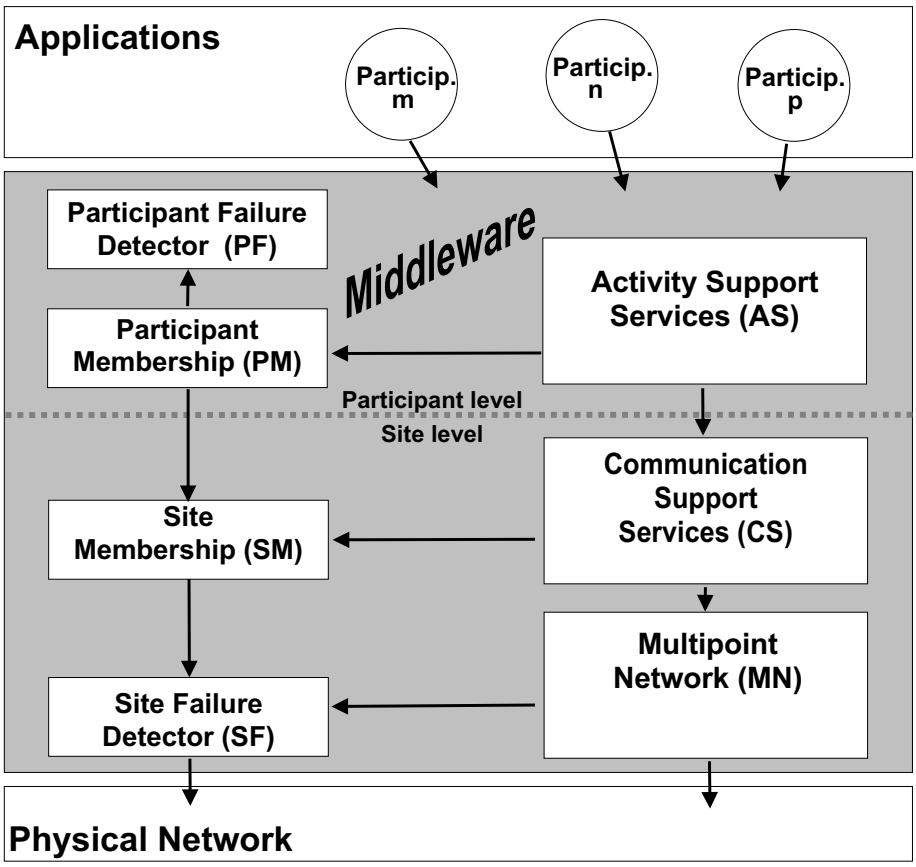

Figure 2.1: Architecture of a MAFTIA Host

module, SM, depends on information given by the SF module. It creates and modifies the membership (registered members) and the view (currently active, or non-failed, or trusted members) of sets of sites, which we call site-groups. The Communication Support Services module, CS, implements cryptographic group communication primitives such as Byzantine agreement and group communication with several reliability and ordering guarantees, clock synchronization, and other core services. The CS module depends on information given by the SM module about the composition of the groups, and on the MN module to access the network.

In the participant level, the Participant Failure Detector module, PF, assesses the liveness and correctness of all local participants, based on local information provided by sensors in the operating system support. The Participant Membership module, PM, performs similar operations as the SM, on the membership and view of participant groups. The PM module monitors all groups with local members, depending on information propagated by the SM and by the PF modules, and operating cooperatively with the corresponding modules in the concerned remote sites. The Activity Support Services module, AS, implements building blocks that assist participant activity, such as replication management,

participant group key management, transactional management, and so forth. Note that several participant groups, or simply groups, may exist in a single site. The site-participant clustering and separation of concerns stated earlier is thus implemented by making a separation between groups of participants (performing distributed activities), and site-groups of the sites where those participants reside (performing reliable communication on behalf of the latter). Clustering can be further enhanced by mapping more than one group onto the same site-group, in what are called lightweight groups [81]. It depends on the services provided by the CS module, and on the membership information provided by the PM module.

Consequentially, the protocols implementing the layers described above all share the topology awareness property. As such, they may run differently depending on their position in the topology, although this happens transparently. For example, the SF protocol instantiated at the Facility Gateways may wish to aggregate all liveness/failure information from the site it oversees, and gather that same information from the corresponding remote Facility Gateways. These considerations may obviously be extended to topology-aware attack and intrusion detection. In conclusion, in MAFTIA we intend to support different styles of basic interactions among the participants, namely: client-server, for service invocations; multipeer, for interactions amongst peers; dissemination, of information in push or pull form; transactions, for encapsulation of multiple actions. The middleware platform provides several services at different levels of abstraction, which contribute to support the above-mentioned interaction styles. At the Communication Services level: basic distributed cryptography, random number and key generation; byzantine agreement; ordered and reliable multicast and their cryptographic variants; time and clock synchronization; etc. Likewise, Activity Services are those concerning the direct support of distributed applications, and from the several possible classes of building blocks supported by the MAFTIA middleware, we currently envision: distributed and replicated state machine support; distributed transaction support; participant key management.

2.2 Site Level

2.2.1 Multipoint Network

Seen from the network, an instantiation of the MAFTIA middleware runs over what we call a Domain, which is the set of sites, spread over the global network, on which any of the distributed applications in question will run. This concept is necessary in order to keep track of all the sites that compose a "MAFTIA system", amongst the immense collection of hosts of the Internet.

Sites are administratively *registered* in the Domain, which is a flat concept, although

the hierarchy and clustering principles of MAFTIA may be used to make the handling and addressing of sites more efficient. Each local Site Failure Detector has a view of the currently reachable and correct sites in the Domain, the *Domain View*, C_{sp}. The view is updated by the failure detection protocols. The Domain may obviously be very large, depending on the scale of the applications. In consequence, the scope of inter-site communication for failure detection may be subdivided using the Facility-based hierarchy: the SF modules of Facility Gateways perform direct intra-facility failure detection, and share failure information with peer Facility Gateway SFs. This topology awareness suits the protocol functionality, but is kept transparent to its semantics.

Multipoint Addressing

The Multipoint Network presents two classes of interfaces: *LocalNet* and *GlobalNet*. The *GlobalNet* is the inter-Facility communication interface and the *LocalNet* is the intra-Facility communication interface. In order to preserve the maximum generality possible, the hierarchical nature of our network model is only made visible when necessary, for example when important performance or security gains can be achieved. This will be exemplified later in this section, when discussing topology-aware services. Transparency is achieved by the logical addressing structure. MAFTIA middleware is built on top of current network architectures, for example, but not limited to: bridged Ethernet for the LocalNet; the Internet for the GlobalNet. The LocalNet and GlobalNet protocols are implemented on top of the relevant standard protocol drivers (IP, UDP, etc.), and traffic is routed to/from either by a dispatcher. A possible implementation of GlobalNet addressing is through an internet group address such as multicast-IP, addressing all the Facility Gateways holding addressed sites. A possible implementation of LocalNet addressing is through a physical group address such as Ethernet multicast.

A logical address A represents a set of addressee sites S. The meaning of the logical address depends on the place where it is handled:

- *in a Facility* - all sites of S in this Facility, plus the Facility Gateway, are addressed; a possible implementation of A is through a physical group address A_L (e.g., Ethernet multicast address);
- *in the Global Network* - all Facility Gateways of Facilities holding sites belonging to S are addressed; a possible implementation of A is through an internet group address A_G (e.g., multicast-IP) addressing all the Facility Gateways holding sites in S;
- *at a Facility Gateway* - a message arriving at a Facility Gateway, coming from the Global Network, with address A, is retransmitted to the Facility, to address A (a

possible implementation is having A_G translated into A_L); a message arriving at a Facility Gateway, coming from the Facility, with address A, is retransmitted to the Global Network, to address A (a possible implementation is having A_L translated into A_G).

The functionality of a Multipoint Network module is exemplified in [42], although that work considers a benign failure model. The following functionality was provided:

- *GlobalNet* - logical addressing (translates to multiple point-to-point routes); FIFO or unordered communication; reliability at this level is just best-effort-1, k-retry, that is, a transmission is tried up to $k + 1$ times, in a best effort to ensure that at least one recipient gets it; rate-based flow control; dynamic rerouting for partition healing; datagram fragmentation to the minimum MTU of inter-Facility routes.
- *LocalNet* - logical addressing (translates to physical multicast); reliability is datagram (no check); rate-based flow control.

Best-effort Delivery

The Multipoint Network abstraction provides a best-effort communication semantics, i.e., the service provided by this module is a best-effort delivery of messages. The reliability of delivery can be improved only by simple algorithms such as retries. Reliable delivery, causal order, total order, and other communication semantics are handled by the Communication Support module (see the following section).

Routing

Routing is a standard function of Internet protocols. Routing is updated whenever link or router failures occur. However, this is not a very prompt and dynamic function in the current Internet. This has effects on system availability when accidental failures occur, and the former will most certainly be amplified if those failures are caused intentionally, i.e. through denial of service attacks.

Inasmuch as effectiveness of distributed system operation has been enhanced with failure detectors, this information can also be used to improve the operation of Internet routing protocols, provided that it does undermine their standard nature. We follow that approach in MAFTIA, namely in what regards quick reaction to partitioning.

Partitioning occurs when a network is split such that sites cannot continue to enjoy the symmetry, transitivity and connectivity properties. Telling network partitioning from site failure is useful, because the system support can do things in order to repair the problem, or avoid a premature declaration of failure of the out-partition sites:

- the simplest is for the Site Failure Detector (SF) to wait, during a given timeout, for a possible merge, instead of immediately declaring failure of the affected sites. This has the virtue of preventing the application from reconfiguring twice (back and forth) when the partition occurs for a short period of time.
- more sophisticated, but quite effective, is partition healing. Besides preventing premature failure declaration as well, it also attempts at removing the physical symptoms of partition.

For a site S_v, a site S_p is partitioned if at least one of the following predicates hold:

- *Non-transitive Partition* - there is a site S_m, such that: S_v is alive for S_m, and S_p is alive for S_m, and S_p is failed for S_v
- *Non-symmetrical Partition* - S_p is failed for S_v, and S_v is alive for S_p
- *Total Partition* - there is a partition P_{out}, to which S_p belongs, and there is a partition P_{in}, to which S_v belongs, such that: all sites in P_{out} are failed for all sites in P_{in}, and all sites in P_{in} are alive for all sites in P_{in}

A non-transitive partition can cause an undesirable membership instability if not handled properly.

The above notion of partition can be extended to several sets of partitions. When the SF has additional information on the state of the network, provided by the MN for example, it may be possible to correlate failure of Global Network links to the defaulting route, in case of partial partition, or to the composition of P_{out}, in the case of total partition. This is not only possible but straightforward to implement in the Internet. Whilst IPv6 is bound to improve routing responsiveness, connectivity glitches may last longer than expected, for example due to the load queues that build up[54].

As an example of how modules of the architecture can use the run-time environment, the SF can use TTCB services to help improve the assessment of a problem, namely to determine whether it is a partition or a site failure, as it can reliable detect the failure of sites by detecting failure of the local TTCB.

Partition healing is attempted whenever the symptoms indicate partitioning. Rerouting of traffic for the partitioned sites is attempted, by resorting to the help of "friendly" sites in the domain.

Partitions are sometimes due to temporary flickering of the network, and thus short-lived. However, most of the times, they result from failure of a link, in a situation where there are alternate paths. However, for several reasons, routing in most global network settings, such as the Internet, is rather static, and not bound to change on account of these occurrences. However, the situation tends to change, namely in the forthcoming new Internet protocols. As such, the *partition healing* facility that we provide is modular and can be disabled if and when standard protocols handle partitioning better. It is performed by the Multipoint Network (MN) protocols running over IP, under the control of the SF.

Let us define S_v, the local site, P_{out}, the *out-partition* (that is, the set of sites which S_v cannot communicate directly with), and P_{med}, the mediating sites, which can communicate with both S_v and sites in P_{out}. For example, imagine that *di.fc.ul.pt*, in Portugal (S_v), is cut off from *cornell.edu*, in the U.S.A. (belonging P_{out}), but *newcastle.ac.uk*, in the U.K., (belonging to P_{med}, the mediator set), communicates both with *di.fc.ul.pt* and *cornell.edu*.

The idea is very simple. The SF, upon deciding 'partial-partition' failure, does not inform the SM module, as would be normal. Instead, it makes a call to the MN module asking to reroute all communication from S_v addressed to site S_p in P_{out}, through site S_m in P_{med}. When the partition disappears, the SF module instructs the MN to go back to the old route. However, this only occurs some time after rerouting. This very simple form of hysteresis is used to avoid routing instability.

A final issue is the interaction of partition healing and firewalls. It was made clear that partition healing is made with rerouting traffic during a partition lifetime. Firewalls can be used, e.g., to protect a Facility, and may impede traffic to go in or out through a new route, as that traffic can be considered to be potentially malicious. Several solutions to this problem exist. The easier is to remove the firewall rules that avoid that rerouting to work. However, that is usually undesirable or the rules would not be there in the first place.

Another solution is to allow the component that wants to reroute traffic to change these rules in run-time. However, to allow something to dynamically change firewall rules is dangerous because a hacker can use the same functionality to reconfigure the firewall at his will. Denial-of-service attacks can be especially easy. Even though, this scheme can be used with caution:

- The components that will be allowed to change the firewall rules have to be clearly identified. If only partitions in the Global Network are handled, only the Facility

Gateway will run partition healing protocols and so only this component will need to interact with the firewall(s).

- It cannot be possible to change all rules, but only those pertaining to MAFTIA middleware traffic between the specific Facilities.
- Communication between the component and the firewall has to be secured: mutual authentication has to occur; authenticity (also against message playback) and integrity of messages has to be guaranteed.

Basic Secure Channels and Envelopes

The MN module provides basic secure channels and message envelopes to protect communication between sites. Secure channels are used to protect communication that lasts long enough for the concept of connection to make sense. They are based on shared symmetric keys that are used to authenticate messages, symmetric cryptography is faster than asymmetric. These cryptographic signatures are used to protect the integrity of communication against forgery, modification, replay, reorder and suppression of messages. Messages can also be encrypted in order to guarantee their confidentiality. In general keys to protect the integrity should be different from keys to protect confidentiality of messages. The reason for this is that the integrity of messages stops being a problem a short time after their delivery while confidentiality of data needs, in general, to be kept for a long time after message delivery. The precise meaning of this "short time" and "long time" expressions depends respectively on the communication system and on the application.

As another example of how the TTCB run-time services can be used by middleware modules, note that the shared keys can be established with the help of the TTCB authentication and key establishment services. This support can be combined with the use of a fully fledged Key Distribution Center or agreed upon using a key agreement protocol [6, 5]. These two solutions require the site to have a long-term asymmetric key pair for authentication. The TTCB has such a key pair, and it is able to store it with a high degree of security, by definition of TTCB. Again, this support can be combined with the use of a fully fledged Certification Authority, for provision of new key certificates to newly instantiated MAFTIA host TTCBs, revocation of the same key certificates, renewal, and so forth.

Secure envelopes are used for sporadic transmissions, and seek to achieve site-level transmission security. They resort to per-message security and may use a combination of symmetric and asymmetric cryptography as a form of improving performance, especially for messages with large bodies. With the current state of the art, they can mostly be implemented through the IP security extensions (IPSec)[52].

2.2.2 Communication Support

This section discusses fault-tolerant communication protocols. We focus on protocols subject to arbitrary (Byzantine) faults, such as those resulting from malicious attacks and intrusions.

Most secure protocols rely on cryptographic techniques, so we introduce first a set of distributed cryptography building blocks that will be available. Protocols for reaching Byzantine agreement are another important building block. We then describe fault-tolerant broadcast protocols for group communication with various semantics. They guarantee the safety properties needed for realizing fault-tolerant applications using the state-machine replication paradigm.

Distributed Cryptography

Cryptographic techniques, such as public-key encryption schemes and digital signatures [64], are crucial already for many existing secure services. For distributing trusted services, we also need distributed variants of them from *threshold cryptography* [33].

Threshold cryptographic schemes are non-trivial extensions of the classical concept of secret sharing in cryptography: this allows a group of n parties to share a secret such that t or fewer of them have no information about it, but $t + 1$ or more can uniquely reconstruct it. However, one cannot simply share the secret key of a cryptosystem and reconstruct it for decrypting a message because as soon as a single corrupted party knows the key, the cryptosystem becomes completely insecure and unusable.

A *threshold public-key cryptosystem* looks much like an ordinary public-key cryptosystem with distributed decryption. There is a single public key for encryption, but each party holds a *key share* for decryption (all keys were generated by a trusted dealer). A party may process a decryption request for a particular ciphertext and output a decryption share together with a proof of its validity. Given a ciphertext resulting from encrypting some message and more than t valid decryption shares for that ciphertext, it is easy to recover the message; this property is called *robustness*. The scheme must also be secure against adaptive chosen-ciphertext attacks in order to be useful for all conceivable applications (see [90] for background information). The formal security definition can be found in the literature [92]; essentially, it ensures that the adversary cannot obtain any meaningful information from a ciphertext unless she has obtained a corresponding decryption share from at least one honest party.

In a *threshold signature scheme*, each party holds a *share* of the secret signing key

and may generate shares of signatures on individual messages upon request. The validity of a signature share can be verified for each party. From $t + 1$ valid signature shares, one can generate a digital signature on the message that can later be verified using the single, publicly known signature verification key. In a *robust* and *secure* threshold signature scheme, it is infeasible for a computationally bounded adversary (1) to produce $t + 1$ valid signature shares that cannot be combined to a valid signature, and (2) to output a valid signature on a message for which *no* honest party generated a signature share.

Another important cryptographic primitive is a *threshold coin-tossing scheme*, as used by the randomized Byzantine agreement protocols. It provides arbitrarily many unpredictable random bits that can be accessed in arbitrary order. Intuitively, one may imagine that a coin-tossing scheme maps an arbitrary bit string—the name of a coin—to a random bit, which is unpredictable before not some honest parties reveal corresponding information. We have developed the first practical cryptographic scheme for coin-tossing recently; its security is based on the Diffie-Hellman problem [18]. It will be used for MAFTIA agreement protocols.

A major complication for adopting threshold cryptography to our partially synchronous system is that many early protocols are not robust and that most protocols rely heavily on synchronous broadcast channels [33]. Only very recently, non-interactive schemes have been developed that satisfy the appropriate notions of security, like the threshold cryptosystem of Shoup and Gennaro [92] and the threshold signature scheme of Shoup [91]. Both have non-interactive variants that integrate well into asynchronous settings, which we also intend to support. However, they can be proved secure only in the so-called *random oracle model* that makes an idealizing assumption about cryptographic hash functions [10]. This falls short from a proof in the real world but gives very strong heuristic evidence for their security; there are many practical cryptographic algorithms with proofs only in this model.

Byzantine Agreement

Byzantine agreement requires all parties to agree on a binary value that was proposed by an honest party. The protocol of Cachin et al. [18] follows the basic structure of all randomized solutions (e.g., [11]) and guarantees termination within a constant expected number of asynchronous rounds, except with negligibly small probability. It achieves the optimal resilience $n > 3t$ by using a robust threshold coin-tossing protocol, whose security is based on the so-called Diffie-Hellman problem. It requires a trusted dealer for setup, but can process an arbitrary number of independent agreements afterwards. Threshold signatures are further employed to decrease all messages to a constant size. As mentioned before, its security proof relies on the random oracle model.

A useful primitive is also *multi-valued Byzantine agreement*, which provides agreement on values from larger domains. Multi-valued agreement requires a non-trivial extension of the binary Byzantine agreement protocols mentioned above. The difficulty in multi-valued Byzantine agreement is how to ensure the "validity" of the resulting value. One approach to this is an "external validity" condition, using a predicate with which every honest party can determine the validity of a proposed value. The protocol must then guarantee that the system may only decide for a value if it is acceptable to honest parties.

Group Communication

Simulating the abstraction of a broadcast channel on top of a network of point-to-point channels with process failures is a difficult task. For systems where parties (processes) can crash, the problem is quite well understood today, after a considerable research effort was made during the last 20 years. It has led to a set of established notions and protocols for reliable broadcast with FIFO, causal, and total ordering properties. We refer to the survey of Hadzilacos and Toueg [46] for a comprehensive treatment of this area.

For systems with malicious faults, though, there are still several open problems in terms of definitions as well as protocols. The investigation of these problems is one of the goals of MAFTIA.

A basic broadcast protocol in a distributed system with failures is *reliable broadcast*, which provides a way for a party to send a message to all other parties. Its specification requires that all honest parties deliver the same set of messages and that this set includes all messages broadcast by honest parties. However, it makes no assumptions if the sender of a message is corrupted and does not guarantee anything about the order in which messages are delivered. The basic reliable broadcast protocol of our architecture is an optimized variant of the elegant protocol by Bracha and Toueg [15]. A variation of it, called *consistent broadcast*, is also foreseen, since it is advantageous in certain situations. It guarantees uniqueness of the delivered message (thus the name consistent broadcast), but relaxes the requirement that all honest parties actually deliver the message—a party may still learn about the existence of the message by other means and ask for it. A similar protocol has also been used by Reiter [78].

An *atomic broadcast* guarantees a total order on messages such that honest parties deliver all messages in the same order. Any implementation of it must implicitly reach agreement whether or not to deliver a message sent by a corrupted party and, intuitively, this is where Byzantine agreement is needed. The basic structure of the atomic broadcast protocol follows the atomic broadcast protocol of Chandra and Toueg [25] for the crash-failure model: The parties proceed in global rounds and agree on a set of messages to

deliver at the end of each round.

At the beginning of a round, each party proposes to deliver the messages it knows using a variation of reliable broadcast. Multi-valued Byzantine agreement is then used to determine a party who has correctly broadcast a valid proposal. All messages contained in the selected proposal are delivered according to a fixed order. This atomic broadcast protocol guarantees liveness, i.e., a message broadcast by an honest party cannot be delayed arbitrarily by the adversary once it is known to at least $t + 1$ parties.

Atomic broadcast and multicast protocols (also known as total order) have proved to be extremely useful in supporting many fault-tolerant distributed applications. For instance, total delivery order is a requirement for the implementation of replicated state-machines [86], which is a general paradigm for implementing fault-tolerant distributed applications.

Although several protocols have been described in the literature [4, 16, 13, 26, 36, 50, 51, 55, 62, 63, 71], few were specifically targeted to operate in (geographically) large-scale systems. In a large scale network processes' traffic patterns are usually heterogeneous. The same applies to the network links: some processes will be located within the same local area network whereas others will be connected through slow links, and thus subject to long delays. In such an environment, none of the previous approaches can provide optimal performance.

The topology-aware total order protocol recognizes that some ordering mechanisms are more appropriate for local-area networks and other more suitable for the wide area network. Since we are targeting a WAN-of-LANs network model, we have designed an hybrid scheme, where each process is able to operate with the ordering mechanism that is most suitable given its position with regard to its peers in the network topology. If all processes are in the same cluster (single or set of interconnected LANs), one mechanism is used. If all processes are in different clusters, another mechanism is used. In intermediate scenarios, different mechanisms are integrated in a hybrid protocol [80].

A *secure causal atomic broadcast* is an atomic broadcast that also ensures a causal order among all broadcast messages, as put forward by Reiter and Birman [79]. Causal communication enforces a logical precedence between messages [16]: *Logical precedence* in a distributed system, in which information is exchanged only by transmitting messages, a message m is said to *precede* or to be *potentially causally related* to a message n, represented as $m \rightarrow n$, only if: (i) m and n were sent by the same process and n was sent after m or; (ii) m has been delivered to the emitter of n before n was sent or; (iii) there *exists* x such that $m \rightarrow x$ and $x \rightarrow n$. With arbitrary faults causality can already be violated by any process that merely observes a message, say during an agreement protocol. Protecting agains this requires encryption.

Experience has shown [85, 16, 71, 96] that the design of distributed applications can be simplified if messages are received in order of logical precedence. Since extra complexity would be added to such applications, should the communication subsystem not provide causal delivery, several algorithms have been proposed to implement this ordering discipline [56, 88, 16, 40, 71, 55]. Nevertheless, despite its advantages, the use of causal communication has been somehow limited by the overhead incurred by existing implementations. A major cost of protocols that preserve logical precedence is the size of "history" information that needs to be stored and exchanged to maintain causality, specially in large-scale systems where group addressing is used.

A topology-aware approach can benefit from the WAN-of-LANs model, allowing to extend previous results on causal history compression using knowledge on the topology of the communication structure. A compression technique based on the concept of a *causal separator*, a set of nodes of the communication graph that can be used to filter causal information is presented in [83].

A secure causal atomic broadcast can be implemented by combining an atomic broadcast protocol that tolerates a Byzantine adversary with a robust threshold cryptosystem. Encryption ensures that messages remain secret up to the moment at which they are guaranteed to be delivered. Thus, client requests to a trusted service using this broadcast remain confidential until they are scheduled and answered by the service. The threshold cryptosystem must be secure against adaptive chosen-ciphertext attacks to prevent the adversary from submitting any related message for delivery, which would violate causality in our context. Maintaining causality is crucial in the asynchronous environment for replicating services that involve confidential data.

Time and Clock Synchronisation

In order to implement synchronous and partially synchronous protocols, we need time and clock synchronization primitives for those hosts with no local TTCB is available. Several clock synchronization protocols are well established today [105, 53, 93]. However, most if not all of them assume only benign faults.

The MAFTIA model suggests to use topology-aware clock synchronization, which can exploit the WAN-of-LANs network model and also tolerates malicious faults. More precisely:

- At the LAN level, a protocol tailored to local area networks is used. The protocol fully exploits the intrinsic attributes of these networks: error rate is low, transmission delay is bounded but with high variance, median transmission delay is close to the

minimum, and message reception is *tight* in absence of errors, meaning that the low-level message reception signal occurs at approximately the same time in all nodes that receive it. This feature can be made fully deterministic when operating from real-time kernels. It is a crucial feature for the mechanism underlying the synchronization algorithm.

- At the WAN level, a GPSs satellite system or any other global infrastructure can be used as the "global network" link between local networks.

The integrated solution combines the LAN-level algorithm with the WAN-level service in a hierarchical manner. This solution can be implemented on virtually any large-scale distributed computing infrastructure as we see them today—such as the wide-area point-to-point Internet.

Such a solution is presented in [98] for a system with a benign failure mode. An adaptation for a malicious environment requires the use of several security mechanisms. Communication has to be secure: message integrity and authenticity. The correctness of sites with clocks has to be assessed in order to avoid a malicious clock from desynchronizing others.

A clock synchronization protocol for a homogeneous network that tolerates malicious faults has been presented by Barak et al. [8]. It relies on a completely connected network with links between any two processors, and can withstand up to one third of faulty processors at any time. Through the use of *proactive secure* mechanisms, an arbitrary number of faults is actually tolerated over the whole lifetime of the system.

Such solutions can be used in MAFTIA to synchronize the clocks of hosts without a local TTCB. For hosts with a local TTCB, this component synchronizes its clock with other local TTCBs and gives a Trusted Absolute Timestamping that can be used reliably by hosts. The synchronism and security of the TTCB make the internal implementation of clock synchronization algorithms simple.

2.3 Participant Level

2.3.1 Activity Support

Replication management

One of the goals of the MAFTIA middleware architecture is to provide fault-tolerant services to clients for applications. The approach pursued here is to base fault-tolerance

on replication. That is, the service is not offered by a single server but by a group of servers, which offer the service collaboratively. From a clients' point of view, the replicated service should behave just as if it were implemented by a single server. Achieving this is highly non-trivial, in particular if some of the replicated servers are subject to crashes or malicious attacks.

In general, one can distinguish between two forms of replication management: *passive* and *active*.

In a system with *passive* replication, there is a distinguished server among the group of servers, who acts as the "master" or primary server, and executes all operations. It receives a request from a client, processes it, and answers to the client. All the other servers (the "slaves" or replicas) merely mirror the operations of the master, and do so only when told by the master. This ensures that all replicas have the same internal state as the primary and can take over as a new primary at any moment, should the primary fail. This form of service replication works for arbitrary operations, including non-deterministic ones.

The difficulty with passive replication is the transition from one primary to the next, because all servers need to agree on who the next primary is and which requests have already been processed. Once this agreement is reached, the new primary continues with processing the next available request. Additional problems arise should the primary be corrupted and modify an honest client's request. The replicas need some way to verify the primary's operations in this case, otherwise this approach cannot be used.

With *active* replication, all servers reach agreement on the requests to execute before actually processing them. Clients are supposed to send their requests to all servers and also receive answers from all servers; they can determine the result of the operation by majority voting. This is usually called the "state machine replication paradigm" [86]. It works only for services with deterministic operations, as otherwise the servers would not be able to maintain the same state. Active replication requires that client requests are delivered by the servers in a global order, which can be ensured by an disseminating client requests using an atomic broadcast protocol.

Both forms of replication can be used in systems with arbitrary malicious faults. For example, Castro and Liskov describe a system based on passive replication [24]. The approach of Reiter and Birman [79], the protocols of Doudou, Garbinato, and Guerraoui [37], and the approach sketched in Section 2.2.2 are based on active replication.

Key Management

All cryptographically secure protocols and services discussed above require some key material to be present at each party.

In order to identify a party in the system, it is associated with the public key of a signature scheme, for which the party must know the corresponding signing key. The association of names to public keys can either be made once and for all in the initialization data, or it can be done dynamically by a certificate using the system manager's public key. This key has then to be known to all parties.

Once such a public-key infrastructure is in place, a symmetric key between each pair of parties can be generated using standard cryptographic techniques: they establish a secure session, one of them chooses a fresh authentication key, and sends it to the partner over the encrypted link. This symmetric key can be used for authenticating the messages between those parties using a MAC (message authentication code).

Keys for the threshold-cryptography protocols are needed as well. Since the individual key shares, which a party holds, are depending on the global public key of the schemes, they cannot be generated individually, like the pairwise authentication keys. They must either be generated by a trusted dealer or by a secure protocol among all parties.

Using a trusted dealer for initializing every party, of course, simplifies the setup of the system. Unfortunately, it also introduces a single point of failure and a prime target for an attacker. However, a trusted dealer for initializing the system may be the only feasible way of distributing the secret key material. If the system uses dynamic groups and new parties may join a group at any time during operation, the dealer is needed continuously, creating another undesirable vulnerability. If the system uses static groups, all information available to the dealer should be destroyed as soon as the system is running.

There are protocols for distributed generation of the keys for threshold cryptosystems and signature schemes in the cryptographic literature [43]. However, they are much less practical than the threshold cryptosystems themselves because all protocols known today rely on synchronous broadcast channels and are considerably more expensive than the operations of the cryptosystems. True, broadcast channels can be simulated using Byzantine agreement, and synchrony could be added using a trusted time service. But the resulting protocols would not be very efficient and a trusted dealer might still be preferred. From the security point of view, the dealer is only needed once for initialization, after all.

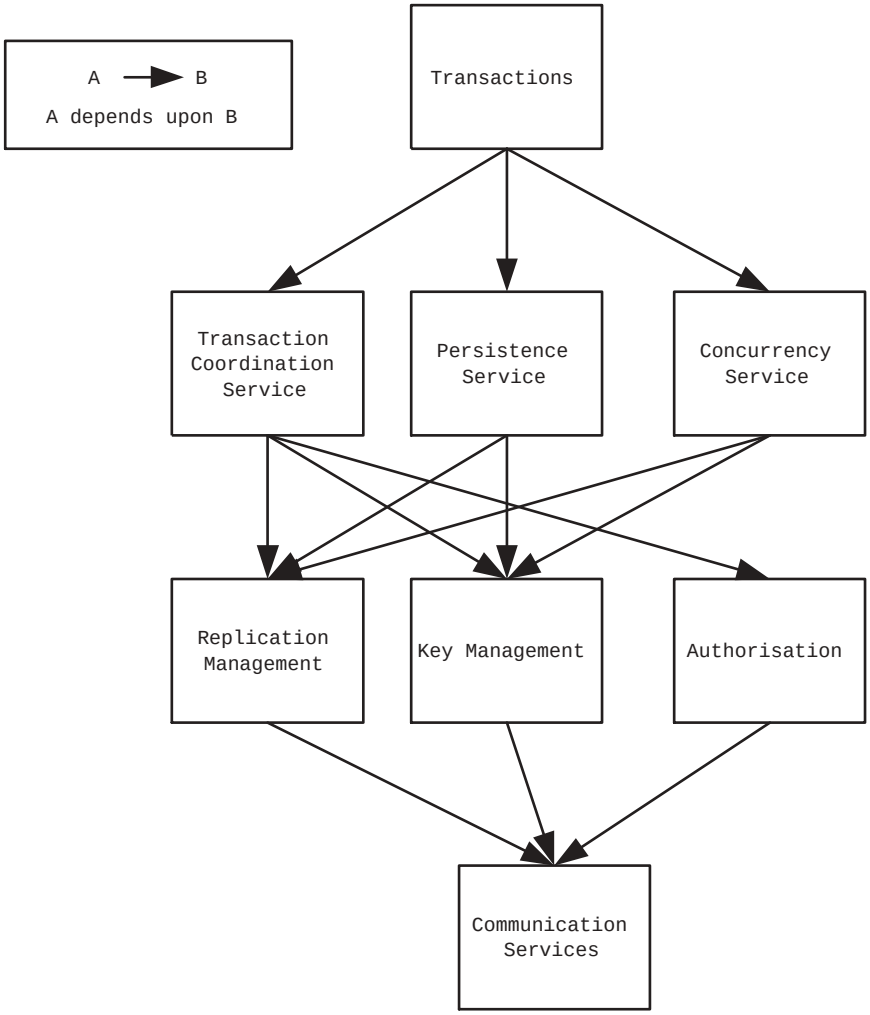

Figure 2.2: Transactional Management Service Architecture

Transactional Management

The transactional management service will provide transactions that are resilient to intrusions, i.e. intrusion-tolerant transactions. The transactions will have reliable, secure and exactly-once behaviour. The case studies have pointed out the need for such transactions, for example the Tradezone case study describes a B2B[1] application that requires intrusion-tolerant transactions in order to support the reliable and secure ordering of goods from suppliers across open networks.

Initially the focus will be on tolerating outsider intrusions rather than insider intrusions. An outsider intrusion is an unauthorised increase in privilege, i.e. a change in the

[1] business-to-business

privilege of a user that is not permitted by the system's security policy. An insider intrusion is an abuse of privilege or misfeasance, i.e. an improper use of authorised operations. For example, we want to prevent outsiders interacting with the transaction service in a way for which they do not hold sufficient privilege. Later, we intend to examine the problems of making multi-party transactions tolerant to outsider intrusions, and also explore how insider intrusions can be tolerated through the use of Coordinated atomic actions (CA actions).

We have decomposed the transactional management service into multiple services in the style of the CORBA object transaction service [70]. The architecture of the transactional management service is represented in Figure 2.2 in which dependence relations between the services are depicted. The "functional" services shown are: the transaction coordination service, persistence service, and concurrency service. Other services such as the replication management service, key management service, authorisation service and communication services (including group communication, and Byzantine agreement) are assumed to be already provided in the MAFTA middleware architecture and are described elsewhere.

The *transaction coordination* service is a protocol engine that implements the protocols for transactions, it drives all the other services in the implementation of transactions. Besides implementing traditional transactional protocols it will ensure that only authorised participants may interact with the transactional management service. The *persistence* service makes durable the state of resources involved in the transactions, and also the state of the transaction itself. The *concurrency* service manages the isolation of resources by enforcing a locking scheme on them. This protects resources from participants who are not members of the same transaction as the resources.

The *replication management* service, *key management* service, *authorisation* service and *communication services* will be used to make the other services intrusion-tolerant. Critical service components will be replicated which will increase the availability of the services. Group communication and multi-valued byzantine agreement will be used to build services that can tolerate the subversion of a certain proportion of replicated components. Key management will be used to manage keys that will be used to establish secure channels between participants, services and resources and authenticate participants. The authorisation service will be used to determine whether participants are authorised to participate in certain transactions.

Once we have a good understanding of the issues surrounding implementing transactions that tolerate outsider intrusions, we will investigate making Coordinated atomic actions (CA actions) [106][107] tolerant to outsider intrusions. CA actions support multi-party interactions and provide the capability to tolerate application-level faults as well as reported unmasked low-level faults from the underlying system. We will require support

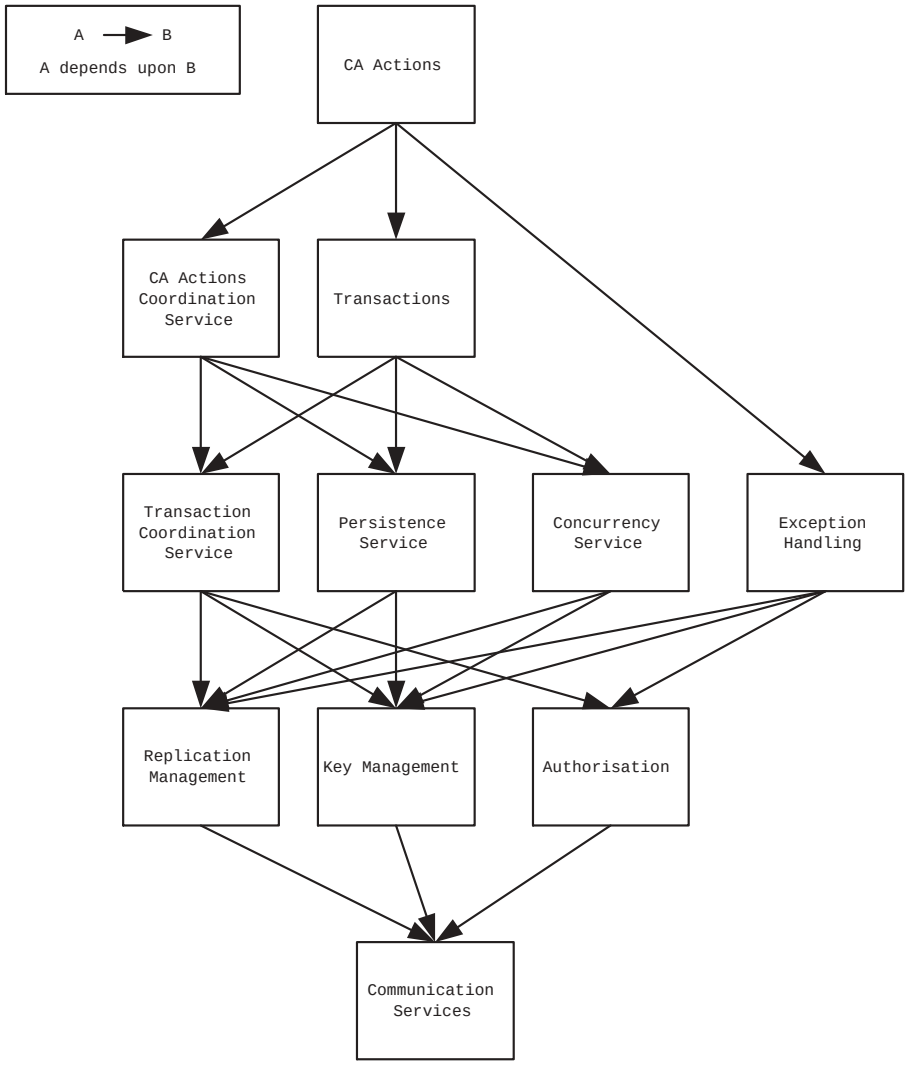

Figure 2.3: Extended Transactional Management Service Architecture

for multi-party transactions in order to support more advanced application models where multiple participants cooperate with each other and compete with external participants for access to resources. For example, as in the telemedicine case study where multiple participants come together to carry out a diagnosis and interact with both shared and global resources.

CA actions will be implemented by an extended transaction management service. This service will build on top of the initial transactional management service. The revised architecture is shown in Figure 2.3. Intrusion-tolerant transactions will be used in order to support competitive concurrency, and additional services will be introduced in order to support the ability to deal with application-level faults (an exception handling service), and coordinate the CA actions protocols (a CA actions coordination service).

The *CA actions coordination* service is a protocol engine that implements the protocols for CA actions, it drives all the other services in the implementation of CA actions. The *exception handling* service implements exception handling for distributed and concurrent exceptions. Such a service is required in order to support backward and forward error recovery. The protocol used to handle distributed and concurrent exceptions will be based on work in [108].

Again we propose using similar techniques as discussed in relation to intrusion-tolerant transactions to make intrusion-tolerant CA actions.

We also intend to examine the problem of insider intrusion-tolerance. Insider intrusions tend to result from attacks at the application level rather than from attacks that take place below the level of the application. We believe that one approach to tackling this problem is to define a boundary around insider actions and control information flows between insiders and other resources and participants. We also require some way to deal with application-level faults and institute backward or forward error recovery in order to compensate for insider intrusions. CA actions give us a framework that is very close to this, they allow boundaries to be drawn around multi-participant interactions and they also have a framework for dealing with application-level faults. This may mean that an intrusion-tolerant CA actions service might be used as a service to implement insider intrusion-tolerance for itself (through recursive application) or for other services. This will the focus of further work.

2.3.2 Example Usecases

We describe three possible applications of the middleware to develop some sophisticated services. These are all services that must provide high security guarantees and maintain safety and liveness in adversarial environments. In turn, they may be used to build, or to support the execution, of distributed end-user applications. As an important note, the definition of building blocks for the middleware Activity Support Services is not a closed issue. As classes of functions or applications gain importance, to the point of justifying the definition of a set of support services, these may be included in the AS module, in order to provide support to applications of that class.

The usecases discussed here are:

- a certification authority and directory service,
- an authentication service, and
- a digital notary service.

These and other trusted third-party services are described in more detail in a companion MAFTIA deliverable. A fully fledged authorisation service will be described in a forthcoming deliverable.

The distributed trusted services are based on secure state machine replication in the Byzantine model (following [86, 79]). Requests to a particular service are delivered by the broadcast protocols mentioned in Section 2.2.2. A broadcast is started when the client sends a message containing the request to a sufficient number of servers. In general, the client must send the request to more than t servers or a corrupt server could simply ignore the message; alternatively, one could postulate that one server acts as a gateway to relay the request to all servers and leave it to the client to resend its message if it receives no answer within the expected time.

Depending on whether it needs to maintain causality among client requests, a service may use atomic broadcast directly or secure causal atomic broadcast otherwise. If the client requests commute, reliable broadcast suffices.

Each server returns a partial answer to the client, who must wait for at least $2t + 1$ values before determining the proper answer by majority vote. Since atomic broadcast guarantees that all servers process the same sequence of requests, the client will obtain the same answer from all honest servers. If the application returns a digital signature, the answers may contain signature shares from which the client can recover a threshold signature.

Certification Authority and Directory Service

A *certification authority* (CA) is a service run by a trusted organization that verifies and confirms the validity of public keys. It is the crucial element of every public-key infrastructure (PKI). The issued *certificates* usually also confirm that the real-world user defined in a certificate is in control of the corresponding private key. A certificate is simply a digital signature under the CA's private signing key on the public key and the identity (ID) claimed by the user.

The CA has published its own public key of a digital signature scheme. When a user wants to obtain a certificate for his public key, he sends it together with his ID and credentials to the CA. The ID might consist of name, address, email, date of birth, and other data to uniquely identify the holder. Then the CA verifies the credentials, produces a certificate if they pass, and sends the answer back to the user. The user can verify his certificate with the public key of the CA. For its certificates to be meaningful, the CA must have a clearly stated and publicized policy that it follows for validating public keys and IDs; this policy might change over time.

A *secure directory* service maintains a database of entries, processes lookup queries, and returns the answers authenticated by a signature under its private signing key. The corresponding signature verification key is available to all clients. Several examples of secure directories exist in distributed systems today and more are needed in the future, such as authentication for the Internet's domain name system.

Internally, a secure directory works much like a CA: it receives a query, retrieves some values from the stored database, generates a digital signature on the result, and sends both back to the client. Additional functionality is needed for updating the database.

Both services can be implemented in our distributed system architecture. Requests must be delivered by atomic broadcast to ensure that all servers return the same answers. Updates to the database must be treated in the same way. The digital signature scheme of the service is replaced by the corresponding threshold signature scheme, which requires minimal changes to the clients in the case of [91]. In the server code, computing the digital signature is replaced by generating a signature share.

Note that atomic broadcast is crucial for delivering any request that changes the global state; only if a CA never changes its policy and all of its certificates are independent of each other does it suffice to use reliable broadcast.

Authentication Service

An *authentication service* verifies the claimed identity of a user or of a process acting on behalf of a user. The user must present secret information that identifies her or carry out a zero-knowledge identification protocol. If verification succeeds, the service will take some action to grant the request, such as to establish a session or return a cryptographic token for later use; this depends on the context in which the service is used. If the answer contains a freshly generated, random session key, as in Kerberos [94], such an authentication server is also called a key distribution center (KDC). Communication between the authentication service and clients may be encrypted and signed with the public key of the service.

The security assumption about the authentication service is that it acts honestly when verifying a password or an identification protocol and never grants a request without having seen the proper identification. The reference data against which the verification occurs is assumed to be public but immutable; this is the case for Unix-style password authentication and for zero-knowledge identification protocols, for instance. But a KDC based on symmetric-key cryptography must also protect the corresponding master secret key.

A distributed authentication service consists of several authentication servers that

are initialized by a trusted dealer and have access to the public reference data. Client requests are distributed by atomic broadcast to all servers. If the request contains sufficient information to authorize the user, the service must produce a suitable cryptographic token.

We distinguish two cases for this, depending on whether the cryptographic token uses public-key techniques or not:

- In a *public-key scenario*, the response of the authentication service is a digitally signed message, which can be thought of as a specialized certificate. Thus, the threshold signature protocols for a CA are used as described in Section 2.3.2.
- If *symmetric-key cryptography* is used, the servers maintain a shared master key and the response consists of an encryption under the master key, just as in Kerberos. An efficient non-interactive protocol to realize a distributed KDC was presented by Naor, Pinkas, and Reingold [69]. The cryptographic mechanism underlying their protocol is in fact the same as used for realizing the distributed coin-tossing scheme in our Byzantine agreement protocol [18] and integrates nicely with our architecture.

As in the general approach, a client can assemble the cryptographic token from the answers of all servers that authorizes her. Existing authorization protocols that use this token require some minimal changes in the cryptographic algorithms.

Notary Service

In its simplest form, a *digital notary service* receives documents, assigns a sequence number to them, and certifies this by its signature. Such a service could, e.g., be used for assigning Internet domain names or registering patent applications. A notary must process requests sequentially and atomically; it updates its internal state for each request.

In many notary applications, the content of a request must remain confidential until the notary processes it in a single atomic step. For example, a competitor in the patent application scenario might try to file a related patent application, have it processed first, and claim the invention for himself.

A distributed notary can be realized readily using our architecture since it involves a simple state machine to be replicated. Client requests must be disseminated by secure causal atomic broadcast to rule out any violation of their confidentiality. For if no encryption were used, a corrupted server could see the contents of the request during atomic broadcast and arrange that the service schedules and processes a related request of the adversary before the original one. The same attack is possible if the cryptosystem is not secure against adaptive chosen-ciphertext attacks.

As the answer of the notary service is a digitally signed message, clients obtain their receipt as described before in the CA example.

3 Verification and Assessment

Tom McCutcheon, William Simmonds, *DERA, Malvern (UK)*
Birgit Pfitzmann, Matthias Schunter, *Universität des Saarlandes (D)*
Michael Waidner, *IBM Zurich Research Lab (CH)*

3.1 Rigorous Security Model

The first objective of verification and assessment in MAFTIA is to develop a formal, mathematical model that allows precise definitions of the notions developed by the other work packages. This objective has largely been achieved now.

3.1.1 General System Model

An essential part of this task was to give a precise model of systems. In this model, we do not yet abstract from anything, because for any later abstraction, we want to rigorously verify that we do not lose anything with respect to the real world. Hence the system model must be probabilistic, and must allow certain security-specific aspects to be expressed, such as security parameters and adversarial scheduling with realistic adversarial information and power.

Such a model was given for the synchronous case in [72, 73] and in a forthcoming report [74] for the asynchronous case. The only abstraction is that the model is digital, i.e., vulnerabilities depending on analog system properties have to be considered separately. For a detailed discussion of the related literature and why new models were needed, see these papers; some related system models are [9, 67, 89, 21].

We now discuss how this rigorous model relates to the system model in Section 1 of this deliverable:

1. Several of the typical fault models from Section 1.1 have been explicitly formalized: fail-stop (benign) and malicious faults, computationally restricted and unrestricted adversaries, threshold and more general adversary structures, and static and dynamic adversaries. The different aspects can be combined arbitrarily. The models are all expressed as general transformations from "intended structures" (the planned error-less system) to "actual structures" and thus hold for all protocols in the MAFTIA architecture.

2. Two types of synchrony (compare Section 1.2) have been formalized, a synchronous model that nevertheless does not (unreasonably) assume that the adversary adheres to any rules about timing, and an asynchronous model. The asynchronous model (although we currently use it to express totally time-free real systems) is somewhat more general in allowing local subscheduling. Partially timed models may be an interesting addition in the future. The crucial protocols between the TTCBs may in fact be expressible in the synchronous model already, except that we are using discrete time.

3. Arbitrary topological models and interaction styles can be expressed in this formalization. However, it may be useful in the future to give naming conventions for certain topologies that occur often.

4. Group models are not visible at all in the general system model because they are virtual concepts; they will turn up as properties of certain specific systems.

3.1.2 Security Notions

Before one can verify a system (with or without faults), one needs a specification, i.e., one has to say what one wants to prove. In security, this may include both what the system should do (integrity and availability) and what the adversary should not learn (confidentiality). The strongest definitions are those that include both aspects. This is done by defining a reference system, often called *ideal system* or *trusted host*, and requiring that the real system is, from the point of view of its users, indistinguishable from this ideal system. Our model extends known cryptographic notions of indistinguishability to the general reactive systems we defined.[1]

Integrity can also be defined as individual properties (e.g., any message that is accepted was previously sent). In the context of distributed protocols, such properties are often formalized in temporal logic (see, e.g., [39]). We have therefore also provided a definition of such integrity properties, extended by cryptographic aspects in [73].

Both styles of specification are adequate for the classes of protocols considered in this deliverable and have been used before, either in a non-cryptographic version or not so rigorously.

As a concrete example belonging to the MAFTIA middleware part, basic secure channels (cf. Section 2.2.1) were specified in this model in the ideal-system style, both synchronously and asynchronously. We believe that the methodology used to make this

[1] However, how to include a form of availability in the asynchronous case is in general still an open problem.

specification is powerful enough to make further specifications quite easily. (We also made one for a higher-layer protocol.) Such specifications correspond to precise API definitions with additional guarantees; hence it may be useful to make several of them in the context of API definitions in the next phase of WP2.

3.1.3 General Architecture Aspects

One of the main aspects of the MAFTIA middleware is layering. The basis for using such a layering also in the verification and assessment is a so-called composition theorem: Given the specification Sys'_0 of a lower-layer system, one wants to design and prove higher-layer protocols Sys_1 based solely on this specification. Now one expects that one can "plug in" any secure real lower-layer system Sys_0 and the higher-layer protocol will still fulfil its own specification.

We have proven such a theorem for all our model variants. No such theorem was known before for reactive cryptographic systems. (A non-reactive composition theorem was proven in [20].) Hence the layered systems can indeed be proven layer-wise just as they are designed, at least as far as ideal-host specifications go. We have also shown that integrity properties shown for the specification are also valid in the real systems.

These theorems are also an important step towards the third objective of verification and assessment in MAFTIA, to investigate how validations made using formal methods and abstractions from cryptographic primitives can be used to deduce security when provably secure real cryptographic primitives are used instead of the abstractions: The ideal-host specifications can be abstract (and are indeed abstract in the examples so far). Hence real systems can be replaced for abstract systems. However, these particular abstractions have not yet been written in CSP, the specific formal method mainly used in MAFTIA.

3.1.4 Verification of Concrete Systems

We have rigorously verified protocols for basic secure channels, using asymmetric encryption and signatures, in both the synchronous and the asynchronous model. This was a particularly rigorous mathematical proof far beyond the usual level of detail in cryptographic proofs, to make sure that all the aspects of the model really fit together. (Another such proof was made for a WP4-like protocol.)

We hope that the proof techniques developed can now be applied to prove further protocols much faster, or one might again omit many of the details.

Besides this specific work within the general model, individual MAFTIA protocols were proved less rigorously by the people inventing them, e.g., those in Section 2.2.2.

3.2 Methods of Verification and Assessment

3.2.1 Use of CSP

This section is intended as a non-technical introduction to, and progress report on, WP6's verification of MAFTIA protocols to date. It is taken from a more detailed technical report that will appear as D4. That report will include full model scripts, documentation, and technical details of CSP usage and modeling practices. For those readers who may not be familiar with CSP, this section has been included as a very brief introduction to the calculus, its usage and its applicability to the verification of large protocols as may be found in WP2.

Communicating Sequential Processes (CSP)[48] is a calculus for describing and analyzing the behaviour of a collection of concurrent interacting processes or systems. The CSP calculus is used to specify the nature of the process interactions – specifically, how communications between the constituent processes affect state and subsequent behaviour. The foundation of the calculus is the notion of engagable 'event' (or 'action') primitive. A process' set of engagable events is termed its 'alphabet'. In the CSP algebra, processes can be 'put in parallel' so that their interaction is through simultaneous engagement in events common to their alphabets[2]

The emergence of automated tools in support of CSP - notably the FDR[59] refinement checker - has elevalated CSP beyond a theoretical, 'academic' manual reasoning language. CSP/FDR is now a widely-used, increasingly popular, formal verification tool supported by a growing and constructive user-community. It can cite successful applications in fields as diverse as cryptography and industrial and military safety-case studies.

FDR can be used to check for anomolous behaviour such as non-determinism, deadlock, and divergence. However, it is more usually used as a refinement checker - i.e. to verify that all behaviours of an 'implementation' process are behaviours of a 'specification' process (or, put another way, that "the implementation meets its spec".)

CSP/FDR is backed by some impressive theoretical results. Arguably the most notable of the more recent theoretical advancements has been in the field of CSP-oriented data independence theory (D.I.)[57]. If a model written in FDR's machine-readable dialect

[2]In the following sections we also refer to processes being 'interleaved'. This means they have no shared events and so cannot communicate with each other and have no 'memory' or 'state' in common.

of CSP, CSP-m, is to be compileable by FDR, then it must be finite-state, and, in particular, the model's data types must be finite types. The data independence theory sometimes allows us to extrapolate the formal verification of a process parameterized by a particular finite instance of a type, to arbitrary instances of that type.

CSP-m is 'untimed-CSP' and it is non-proabalistic – i.e. the CSP-m algebra has no intrinsic notion of either timing or probability. Accordingly, various discrete timing abstractions have arisen in the CSP-user community to become more or less 'standard practice' CSP.

Casper[58] and ProBE[60] are two other notable CSP support tools. The former is a front-end to FDR. It allows the user to specify point-to-point key protocols in a high-level, literate language designed so as to be familiar to researchers in the security-protocol community. Casper generates compileable CSP-m models from the high-level specs. ProBE is probably best described as a 'CSP animator', it allows the user to easily investigate the possible behaviours of a process that would otherwise be determined by the process environment.

One of the issues of CSP/FDR usage that is particularly pertinent to the verification of WP2 protocols is 'state-space explosion'. State-space explosion is a problem that will be familiar to many users of automated model-checking tools, and users of the FDR tool are not excepted. This is explored more fully in Section 3.2.3.

We can expect the major WP2 protocols to be quite complex, and, very possibly, multi-layered protocols that utilize lower-level crypto 'primitives'. Inevitably state-explosion is going to be a big issue in the verification of these protocols. It is likely that we shall have to factor the larger WP2 protocols into smaller, more manageable 'chunks' that are amenable to verification by an automated checker such as FDR. This, of course, raises the question of composition. The Trusted Host theory and its transcription to CSP is certainly very pertinent to this, as is the work on the CSP modeling and analysis of the High Level Architecture (HLA) component integration standard[3].

3.2.2 Selection of MAFTIA Protocols for Verification

The protocol verification work to date has primarily focused on the CSP verification of the Certified Mail, Contract Signing, and Asynchronous Binary Byzantine Agreement (ABBA) protocols. These protocols were selected because of their availability and the fact that they are intrinsic to the MAFTIA architecture.

We found these four protocols technically varied and challenging.

All four protocols provide for 'core services' that are liable to be used directly or indirectly in a MAFTIA environment. The Contract Signing protocols come in both synchronous and asynchronous communications versions. All the protocols were required to be verified for an arbitrary number of parties, transactions, etc. and all can suffer faulty or malicious behavior. Care must be taken in the abstraction from protocols, especially in the abstraction of adversarial behavior.

The verification of the ABBA protocol would be required to address a number of very interesting and challenging 'data independence' and probabilistic issues.

A Byzantine protocol, such as ABBA, is of particular interest to WP6 because the protocol's fault tolerance is in its being able to withstand the possible malfunction or corruption of a particular percentage of the participant parties. In ABBA's case, that percentage is the theoretical maximum, which is less than 33% of the parties. This is a good example of the concept of fault tolerance through redundancy. The verification of ABBA would require original work on the formulation, in CSP, of widely used (N,K,T)-threshold schemes[3].

3.2.3 Discussion of Key Modeling Issues

State-Space Explosion

From the point of view of the CSP user community, the problem of state-space explosion arises when the Labeled Transition System (LTS) representing the CSP-m defined process becomes impracticably large – either too large for FDR to compile in the first place, or too large to perform refinement checks on[4].

State-space explosion was a major issue in the verification of the asynchronous Contract Signing protocol. The asynchrony of the communication medium was modeled by 'intercepting' and 'buffering', for arbitrary lengths of time, the messages sent between parties. If M, say, was the datatype representing the message-space, then our buffer could be in any one of $2^{\mathrm{card(M)}}$ different states at any time. In one of our models of a contract signing session between two correct signatories there were potentially 115 different messages that could be sent between the signatories. On the face of it, this would necessitate the compilation of a CSP process of no less than 2^{115} states. Clearly this was infeasible, and we were faced with the problem of substantially reducing the large state-space without

[3]Such as the (N,2N/3,N/3) signature and coin-tossing schemes employed by ABBA. The CSP abstraction of these schemes would need to be data independent in the number of parties, N, if there was to be any chance of formally verifying the protocol for arbitrary large N.

[4]The problem is usually, but not always, the result of a large degree of non-determinacy.

weakening the verification.

The solution to this was to 're-name' all those messages that we supposed would never be sent in practice to a single 'error message' - thereby reducing the effective message-space to a more managable size (about $\sim 2^{10}$ in the above example). This was done in such a way that if we had falsely constrained the possible message traces, then this would be visible in our refinement tests.

The solution required very few amendments to our original models. Moreover, it could not be criticised for making any a-priori assumptions that we would otherwise have wanted to prove – an invalid supposition as to which messages would be sent would be manifest in the refinement checks. It was very effective in reducing state-space to manageable proportions, and, to date it has proved largely sufficient for our needs.

FDR's Set Handling

FDR's set handling is generally agreed to be one of its less strong features. This is manifest particularly when sets appear as parameters in process definitions, aggravating compilation time, sometimes quite dramatically.

To be fair, it is very debatable whether the FDR tool can be very much improved in its set handling capabilities. Sets, particularly when they appear as process parameters, are usually unavoidably expensive to maintain and manipulate internally[5].

We used a simple – but nonetheless effective – technique to speed-up compilation of the model of the asynchronous Contract Signing protocol. The state-space reduction strategy described in the last section reduced the number of possible states of this model to more manageable proportions, but we were still left with a somewhat lengthy compilation time due to the unstructured set parameter, BUFFER, of cardinality $\sim 2^{10}$. We used FDR's renaming facility to re-specify the process that referred to the single BUFFER parameter as card(BUFFER) 'interleaved' processes each referring to only singleton set parameters.

A few other common modeling techniques were employed to further speed up compilation time. One of these uses re-naming to simplify processes with non-variable parameters (usually some sort of 'node identifier').

[5]There are 2^n possible sets in the elements of a type of order n. The problem stems from the fact that FDR has no means of knowing whether it can 'lazily compile' a process with structured or unstructured sets as parameters. That is, although in some contexts it may be that compilation is only necessary for a small fraction of the 2^n possible sets, FDR cannot know in advance what those sets might be.

Type Abstraction

Abstraction can be a very contentious issue in formal modeling. However, with the type of protocol verification that WP6 is concerned with - as opposed to highly subjective 'real-world' scenarios – abstraction is not so problematic. Here, we are validating well-defined interactive algorithms that manipulate well-defined types.

One of the challenges is to convincingly simplify those types. Whatever we do, we must not over simplify a type to the extent that we falsely constrain control flow that would otherwise have been dependent on the results of operations or predicates performed on that type.

Among the more significant types that WP6 has been required to abstract to date are: crypto-signatures; threshold coin-tossing and signature schemes; transaction identifiers; contract bodies; and mail bodies.

Both Saarland and DERA teams abstracted the contracts, mail bodies and transaction identifiers to simple two-element types. To those readers who may not be so familiar with formal methods, this simplification may appear to be rather naïve, however the simplification is, in fact, quite easily justified by the fact that the only operation or predicate that was ever performed by the protocols on any of these three types was testing for equality.

We had to be slightly more careful when abstracting the crypto-signature schemes. Here we had little choice other than to read "is computationally infeasible to forge a signature" to mean, simply, "is unforgeable". Although this type of simplification is widely acknowledged in the formal methods community, it does exemplify the fundamental difference between the 'finite-state' and 'crypto' approaches to validation[6]. Once we maintained that a crypto-signature scheme was unforgeable, then abstracting from the scheme was reasonably straightforward.

Our models of the contract-signing and certified mail protocols refer to only three parties, i.e. two correctly behaving or 'honest' parties, and one faulty, possibly 'malicious' party. This was because the protocols need, at least, to be verified for two correct parties, and for one correct and one faulty party in the case when the correct party initiated the session, and in the case when the faulty party initiated the session.

The argument that this verification could then be extrapolated to an arbitrary finite number of parties is a familiar one. It relied on our assuming that the signature scheme is unforgeable, and on the fact that the transaction identifiers are unique to a session[7]. In

[6]The Trusted Host theory addresses the problem of bridging the two approaches.

[7]From the verification point of view, this means that we can 're-use' TIDs after verifying that a session was 'safely' conducted.

short, this meant that a multi-party Contract Signing process could be modelled simply as the interleaving of a number of 2-party processes[8].

Modeling Faulty or Malicious Behavior

For the non-ABBA protocols, the modeling of a faulty, potentially malicious, process was really quite simple. We made no distinction between an 'intelligent' and 'unintelligent' adversary – our adversary was free to do anything within certain specific constraints imposed by the protocols.

In practice, this meant that a 'faulty' party F, say, was basically given a 'free run' to input and output at will. The only constraints being that the messages that were output by F could not be 'forgeries'. For the contract-signing protocols, this meant that F was limited to outputting only those messages it could have legitimately 'deduced' from messages previously received (by signing previously received messages any number of times). F's initial message-set (or 'knowledge' set) was the set of all those messages signed only by F.

The problem with this type of adversary abstraction is, again, one of potential state-space explosion. Fortunately, we are largely covered here by the deployment of the state-space reduction strategy as described at the beginning of this section[9].

Issues specific to ABBA

The verification of the ABBA protocol proved, as we expected, to be a very challenging exercise, and has accounted for the bulk of the DERA WP6 team's work effort to date.

There were several good reasons why we expected the ABBA verification to be more problematic than that of the other protocols. Firstly, there is no 'obvious' means by which we can extrapolate a verification of ABBA for some fixed N, N=4[10], say, to the general multi-party case (as we did for the Contract-Signing and Certified Mail protocols as described in 3.2.3). Secondly, one of the properties we want to verify for ABBA is a probability of any particular party deciding 'yes' or 'no' in any particular round – but probability is not naturally visible through refinement testing in the failures-divergences

[8] And that the verification of the multi-party process would be a corollary of the verification of a 2-party process.

[9] In this context, it is a variant of the well-known 'lazy spy' abstraction

[10] Four being a familiar lower threshold for N in many Byzantine Agreement protocols.

model that underlies FDR.

The 'topology' of ABBA is deceptively simple. It is easy to think of a 'proof' of ABBA only in terms of 'numbers of parties', and to forget that there are, in fact, not one, but two, potentially unbounded types. These are the parties P, say, and the rounds R. The topology that we must model, then, is of a fully-connected 'grid' of parties for each round, with the 'nodes' of the grid for round r, say, being mapped one-one to nodes of the grid for round r+1.

Verifying ABBA for 'only' particular values of P, then, would be no trivial matter in itself – as we would still have to account for the fact that the number of rounds that the protocol might run for is unbounded.

The first problem was to model a single round of the ABBA protocol data-independently of P. CSP data independence theory would then allow us to extrapolate the positive result of a refinement test pertaining to a particular finite instantiation of the type P, to arbitrary P.

To appreciate some of the subtleties involved in modeling a single round of ABBA, it is necessary to go into a little detail about how ABBA works. In each round of the ABBA protocol, two ballots are held. In each of these ballots, the parties cast a vote for 'yes', for 'no', or explicitly abstain. An 'honest' party waits until it has received 2N/3 'justified' (or authenticated) votes and 'decides' for yes or no if and when, in some round, those votes are all for one of the two values yes or no. ABBA compels the adversarial parties to abide by the voting rules by the use of two-thirds signature and coin-tossing schemes. All the parties have to 'justify' their voting for a particular value in a ballot by having collected 2N/3 distinct signature or coin shares sent with the votes of a previous ballot. The problem, then, was to model one party's 'collection' of two-thirds of the signature and coin-tossing shares data-independently of P, in particular, this meant without referring to card(P).

We omit details of this modeling. We mention only that it involved the parties non-deterministically selecting a 'recipient' party to whom they would then 'send' their votes. The current 'recipient' would then set-theoretically collect the votes sent to it as they 'came in', discarding every one in three. A special vote of 'NULL' was introduced which allowed a party to indicate that it had no vote to cast as yet. This surprisingly simple abstraction was all that was required in order to model the asynchronicity of the comms medium.

The above model had to be amended to take into account certain 'opportunistic' adversarial behavior in which malfunctioning or corrupted parties could sometimes 'lie' and tell one honest party that they were voting for 'yes', for example, whilst telling another honest party that they were abstaining. This could only happen when the adversary could

establish whether or not a two-thirds majority existed for 'yes' (or for 'no'). The model of this behaviour is really quite simple and succinct, but we omit details here.

A single round of the ABBA protocol was then modeled as: $\| p : P @ NODE(p)$ – i.e. NODE(p) processes in shared parallel indexed over the parties p of P, where the 'NODE(p)' process is a single-round of the protocol from party p's 'point of view'.

Unfortunately this meant that we could not use traditional D.I. theory to establish even single-round properties of the protocol as the theory disallows use of the shared parallel operator (i.e. ||) indexed over the type in question (in this case P)[11]. We can get around this, however, by using recently developed 'Data Independent Induction' theory[32][12]. This said, there are good technical reasons why the application of this theory is not so straightforward if we are to incorporate in the inductive reasoning the model of the adversarial behavior cited above.

Our modeling of the single round of ABBA is a quite accurate, truthful abstraction of the protocol's behavior in practice. The danger now is that the quite complex theory that is necessary to build-up a full model of a run of the protocol in terms of contiguous single-rounds will begin to obfuscate that abstraction. The challenge is not in delivering some form of verifiable model of ABBA, but in delivering a verifiable model that is reasonably accessible and not over abstracted. The model must not suppose too many facts that may be 'obvious' only to those that are more familiar with the niceties of the protocol.

ABBA's probabilistic Fast Convergence property, cited at the beginning of this section, is not amenable to proof by refinement testing as probabilities are not visible in the CSP failures-divergence model. Nevertheless, once one has become reasonably familiar with the ABBA protocol, it is not difficult to see why Fast Convergence holds, and we are confident of being able to transcribe this property in the model.

Correlation between Models and Protocols

In this year's WP6 work, correlation between models and protocols has not been a particularly big issue. The challenge with transcribing Trusted Host based verifications in terms of CSP is, interestingly, to elucidate the mappings of formal state machine specs to

[11] As it would allow the cardinality of the type to be calculated, and this, in turn, could be used surreptitiously to influence control flow.

[12] This theory necessitates formulating a 'SUPERSTATE' process that represents a partially completed view of a round of the protocol. These partially completed SUPERSTATES appear in a family of 'base-case' and 'step-case' refinement tests that build-up to a proof that the shared-parallel process indeed satisfies its spec.

CSP models, rather than correlating a purely 'natural language' spec with a CSP model. Correlation is perhaps more contentious in our verification of the ABBA protocol. In the case of ABBA, we must take care that in inferring the 'obvious' from the paper specification, we do not lose sight of the mechanics of the actual protocol.

We anticipate correlation being a more sensitive issue in future work – where we can expect to have to justify the formal verification of much larger architecture solutions and protocols.

3.2.4 Anticipated Results and Potential Issues

Our verification to date has not revealed any major errors in the protocols. This said, we wish to stress that there is some modeling outstanding.

The verification did reveal a few very minor specification ambiguities and implicit environment assumptions[13]. This could help to tighten future specifications intended for publication.

It is intended to complete the verification and writing-up of the Contract Signing and Certified Mail protocols. This will be reported on in D4.

We will discuss details of ABBA with the protocol authors with the intention of selecting the best form of verification. We will implement this and report on our findings in D4.

Although our final model of ABBA may not be as clear-cut as we originally hoped, we are confident of delivering a reasonably well balanced package that demonstrates the capability of CSP/FDR to validate MAFTIA protocols.

[13] Pertaining to the choice of 'initiating' process and the nature of communications between signatories and TTP.

4 Conclusion

This deliverable presents three main contributions to the specification of the MAFTIA middleware architecture:

- *System Model :* a detailed discussion on the several models upon which the system architecture is conceived. The models that were considered include the fault, synchrony, topological and group models.
- *Architecture :* the building blocks and services of the MAFTIA middleware architecture. The architecture was divided into two levels, the site and participant levels, each containing a set of interdependent layers offering the various services.
- *Verification and Assessment :* an introduction, which will be extended in other deliverables, of a formal security model and methods of verification that will be used to validate the MAFTIA middleware architecture.

In the next deliverable of WP2, D24, this specification will be further refined with the precise specification of the APIs and protocols used to access and implement the various services.

Bibliography

[1] M. Abrams, S. Jajodia, and editors H. Podell. *Information Security*. IEEE Computer Society Press, 1995.

[2] Dominique Alessandri. Using rule-based activity descriptions to evaluate intrusion-detection systems. In H. Debar, L. Me, and S. F. Wu, editors, *Proc. Recent Advances in Intrusion Detection, Third International Workshop (RAID2000)*, volume 1907 of *Lecture Notes in Computer Science*. Springer, 2000.

[3] Robert Allen, David Garlan, and James Ivers. Formal modeling and analysis of the hla component integration standard. Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE-6), November 1998.

[4] Y. Amir, L. Moser, P. Melliar-Smith, D. Agarwal, and P. Ciarfella. Fast message ordering and member-ship using a logical token-passing ring. In *Proceedings of the 13th IEEE International Conference on Distributed Computing Systems (ICDCS-13)*, pages 551–560. IEEE, 1993.

[5] Giuseppe Ateniese, Olivier Chevassut, Damian Hasse, Yongdae Kim, and Gene Tsudik. The design of a group key agreement API. Technical Report TR 99-711, USC Computer Science Dept., September 1999.

[6] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. New multi-party authentication services and key agreement protocols. *IEEE Journal of Selected Areas on Communications*, 18, March 2000.

[7] Özalp Babaoğlu. On the reliability of consensus-based fault-tolerant distributed computing systems. *ACM Transactions on Computer Systems*, 5(3):394–416, November 1987.

[8] Boaz Barak, Shai Halevi, Amir Herzberg, and Dalit Naor. Clock synchronization with faults and recoveries. In *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 133–142, 2000.

[9] Donald Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. Journal of Cryptology 4/2 (1991) 75-122.

[10] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. 1st ACM Conference on Computer and Communications Security*, 1993.

[11] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC)*, 1983.

[12] Piotr Berman and Juan A. Garay. Randomized distributed agreement revisited. In *Proc. 23th International Symposium on Fault-Tolerant Computing (FTCS-23)*, pages 412–419, 1993.

[13] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3), August 1991.

[14] A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, 1984.

[15] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, October 1985.

[16] K. Briman and T. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):46–76, 1987.

[17] K. Briman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, 1991.

[18] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. In *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 123–132, 2000. Full version available from Cryptology ePrint Archive, Report 2000/034, http://eprint.iacr.org/.

[19] R. H. Campbell and B. Randell. Error recovery in asynchronous systems. *IEEE Transactions on Software Engineering*, 12(8):811–826, 1986.

[20] Ran Canetti. Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13/1 (2000) 143-202.

[21] Ran Canetti. Studies in secure multiparty computation and applications. Thesis, Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, June 1995, revised March 1996.

[22] Ran Canetti, Rosario Gennaro, Amir Herzberg, and Dalit Naor. Proactive security: Long-term protection against break-ins. *RSA Laboratories' CryptoBytes*, 3(1), 1997.

[23] Ran Canetti and Tal Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 42–51, 1993.

[24] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Proc. Third Symp. Operating Systems Design and Implementation*, 1999.

[25] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[26] J. Chang and N. Maxemchuck. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2(3), August 1984.

[27] D. Cheriton and W. Zwaenepoel. Distributed process groups in v-kernel. *ACM Transactions on Computer Systems*, 3(2):77–107, 1985.

[28] F. Christian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. In *Proceedings of the 15th IEEE International Symposium on Fault-Tolerant Computing (FTCS-15)*, pages 200–206, Ann Arbour, MI, USA, 1985. IEEE Computer Society Press.

[29] F. Christian and C. Fetzer. The timed asynchronous system model. In *Proceedings of the 28th IEEE International Symposium on Fault-Tolerant Computing (FTCS-28)*, pages 140–149, Munich, Germany, 1998. IEEE Computer Society Press.

[30] Miguel Correia, Paulo Veríssimo, and Nuno Ferreira Neves. The architecture of a secure group communication system based on intrusion tolerance. In *International Workshop on Applied Reliable Group Communication*, Phoenix, Arizona, USA, April 2001.

[31] Ronald Cramer, Ivan B. Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret sharing scheme. In Bart Preneel, editor, *Advances in Cryptology: EUROCRYPT 2000*, volume 1087 of *Lecture Notes in Computer Science*. Springer, 2000.

[32] S.J. Creese and A.W. Roscoe. Verifying an infinite family of inductions simultaneously using data independence and fdr. Oxford University Computing Laboratory.

[33] Yvo Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, 1994.

[34] Yves Deswarte, Laurent Blain, and Jean-Charles Fabre. Intrusion tolerance in distributed computing systems. In *Proc. 1991 IEEE Symposium on Research in Security and Privacy*, pages 110–121, Oakland, California, USA, May 1991.

[35] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronysm needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, 1987.

[36] D. Dolev, S. Kramer, and D. Malki. Early delivery totally ordered multicast in asynchronous environments. In *Digest of Papers, The 23th International Symposium on Fault-Tolerant Computing*, pages 544–553. IEEE, 1993.

[37] Assia Doudou, Benoit Garbinato, and Rachid Guerraoui. Abstractions for devising Byzantine-resilient state machine replication. In *Proc. 19th Symposium on Reliable Distributed Systems (SRDS 2000)*, pages 144–152, 2000.

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

[39] E. Allen Emerson. Temporal and modal logic. in: Jan van Leeuwen (ed.): Handbook of Theoretical Computer Science (Vol. B: Formal Models and Semantics); Elsevier Science Publishers B.V., Amsterdam 1990, 995-1072.

[40] C. Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *Proceedings of the 11th Australian Computer Science Conference*, Australia, 1988.

[41] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[42] Jorge Frazão. Enhancing large-scale communication with failure suspectors and dynamic routing. Master's thesis, Instituto Superior Técnico, Lisboa, Portugal, April 1995. in portuguese.

[43] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure key generation for distrege-log based cryptosystems. In Jacques Stern, editor, *Advances in Cryptology: EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310. Springer, 1999.

[44] Li Gong. A security risk of depending on synchronyzed clocks. *Operating Systems Review*, 26(1):49–53, 1992.

[45] Rachid Guerraoui and André Schiper. Consensus: The big misunderstanding. In *Proc. 6th IEEE Computer Society Workshop on Future Trends in Distributed Computing Systems (FTDCS)*, 1997.

[46] Vassos Hadzilacos and Sam Toueg. Fault-tolerant broadcasts and related problems. In Sape J. Mullender, editor, *Distributed Systems*. ACM Press & Addison-Wesley, New York, 1993. An expanded version appears as Technical Report TR94-1425, Department of Computer Science, Cornell University, Ithaca NY, 1994.

[47] Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multi-party computation. *Journal of Cryptology*, 13(1):31–60, 2000.

[48] C.A.R. Hoare. Communicating sequential processes. Prentice Hall (1985).

[49] Geoff Huston. Next steps for the IP QoS architecture. Internet Draft, August 2000.

[50] T. Johnson and L. Maugis. Two approaches for high concurrency in multicast-based object replicationx. Technical Report 94-041, Department of Computer and Information Sciences, University of Florida, 1994.

[51] M. Kaashoek and A. Tanenbaum. Group communication in the amoeba distributed operating system. In *Proceedings of the 11th International Conference on Distributed Computing Systems (ICDCS-11)*, pages 222–230. IEEE, 1991.

[52] S. Kent and R. Atkinson. Security architecture for the internet protocol. IETF Request for Comments: RFC 2093, November 1998.

[53] Hermann Kopetz and Wilhelm Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, C-36(8):933–940, August 1987.

[54] Craig Labovitz, Abha Ahuja, and Farnam Jahanian. Experimental study of internet stability and backbone failures. In *Proceedings of the 29th International Symposium on Fault- Tolerant Computing (FTCS 29)*, Madison, USA, 1999.

[55] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Lazy replication: Exploiting the semantics of distributed services. MIT/LCS/TR 84, MIT Laboratory for Computer Science, 1990.

[56] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[57] R.S. Lazic. A semantic study of data-independence with applications to the mechanical verification of concurrent systems. r.s. lazic. Oxford University D.Phil thesis, 1999.

[58] G. Lowe. Casper: A compiler for the analysis of security protocols. Proceedings of 1997 IEEE Computer Security Foundations Workshop.

[59] Formal Systems (Europe) Ltd. Failures-divergences refinement. N/A.

[60] Formal Systems (Europe) Ltd. Probe. N/A.

[61] N. A. Lynch. *Atomic Transactions*. Morgan Kaufmann, 1993.

[62] K. Marzullo, S. Armstrong, and A. Freier. Multicast transport protocol. Internet RFC 1301, IETF, 1992.

[63] P. Melliar-Smith, L. Moser, and V. Agrawala. Broadcast protocols for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):17–25, January 1990.

[64] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1997.

[65] Chris Metz. IP QoS: Traveling in first class on the internet. *IEEE Internet Computing*, 3(2), Mar/Apr 1999.

[66] F. Meyer and D. Pradhan. Consensus with dual failure modes. In *Digest of Papers, The 17th International Symposium on Fault-Tolerant Computing*, Pittsburgh-USA, July 1987. IEEE.

[67] Silvio Micali and Phillip Rogaway. Secure computation. Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 392-404.

[68] Louise E. Moser and Peter M. Melliar-Smith. Byzantine-resistant total ordering algorithms. *Information and Computation*, 150:75–111, 1999.

[69] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *Advances in Cryptology: EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*. Springer, 1999.

[70] Object Management Group. *CORBAservices: Common Object Services Specification : OMG Document Number 95-3-31*, March 1995.

[71] L. L. Peterson, N. C. Buchholz, and R. D. Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, 1989.

[72] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Secure reactive systems. IBM Research Report RZ 3206 (#93252) 02/14/2000, IBM Research Division, Zürich, May 2000.

[73] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. 7th ACM Conference on Computer and Communications Security, Athens, November 2000, ACM Press, New York 2000, 245-254.

[74] Birgit Pfitzmann and Michael Waidner. Model for asynchronous reactive systems and its application to secure message transmission. IBM report soon.

[75] D. Powell, D. Seaton, G. Bonn, P. Veríssimo, and F. Waeselynk. The Delta-4 approach to dependability in open distributed computing systems. In *Digest of Papers, The 18th International Symposium on Fault-Tolerant Computing*, Tokyo – Japan, June 1988. IEEE.

[76] David Powell, editor. *Delta-4: A Generic Architecture for Dependable Distributed Processing*. Springer-Verlag, Berlin, Germany, 1991. Research Reports ESPRIT.

[77] Michael O. Rabin. Randomized Byzantine generals. In *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 403–409, 1983.

[78] Michael Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *Proc. 2nd ACM Conference on Computer and Communications Security*, 1994.

[79] Michael K. Reiter and Kenneth P. Birman. How to securely replicate services. *ACM Transactions on Programming Languages and Systems*, 16(3):986–1009, May 1994.

[80] L. Rodrigues, H. Fonseca, and P. Veríssimo. Totally ordered multicast in large-scale systems. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, pages 503–510, Hong Kong, May 1996. IEEE.

[81] L. Rodrigues, K. Guo, A. Sargento, R. Van Renesse, B. Gladeand, P. Veríssimo, and K. Birman. A transparent light-weight group service. In *Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems*, Niagara-on-the-Lake, Canada, October 1996.

[82] L. Rodrigues, E. Siegel, and P. Veríssimo. Replication-transparent remote invocation protocol. In *Proceedings of the 13th IEEE Symposium on Reliable Distributed Systems*, Dana Point (CA), USA, 1994.

[83] L. Rodrigues and P. Veríssimo. Causal separators for large-scale multicast communication. In *Proceedings of the 15th IEEE International Conference on Distributed Computing Systems (ICDCS'15)*, pages 83–91, Vancouver, BC, Canada, May 1995. IEEE.

[84] Luís Rodrigues and Paulo Veríssimo. Topology-aware algorithms for large-scale communication. In S. Krakowiak and S. Shrivastava, editors, *Advances in Distributed Systems*, volume 1752 of *Lecture Notes in Computer Science*, chapter 6, pages 127–156. Springer-Verlang, 2000.

[85] Fred B. Schneider. The state machine approach: A tutorial. Technical Report TR86-800, Cornell University, Computer Science Department, December 1986.

[86] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4), December 1990.

[87] H. Schulzrinne, S. Casner, R. Frederik, and V. Jacobson. Rtp: A trasnport protocol for real-time applications. Internet Network Working Group, Request for Comments (RFC) 1889, January 1196.

[88] A. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. Technical report, Department of Computer Science, University of Kaiserlautern, Germany, 1991.

[89] Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. Concur '94, LNCS 836, Springer-Verlag, Berlin 1994, 481-497.

[90] Victor Shoup. Why chosen ciphertext security matters. Research Report RZ 3076, IBM Research, November 1998.

[91] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology: EUROCRYPT 2000*, volume 1087 of *Lecture Notes in Computer Science*, pages 207–220. Springer, 2000.

[92] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In Kaisa Nyberg, editor, *Advances in Cryptology: EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*. Springer, 1998.

[93] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.

[94] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *Usenix Conference Proceedings*, pages 191–202, March 1988.

[95] Douglas R. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995.

[96] R. van Renesse, K. Birman, R. Cooper, B. Glade, and P. Stephenson. Reliable multicast between micro-kernels. In *Proceedings of the USENIX Workshop on Micro-Kernels and Other Architectures*, pages 269–283, Seattle, Washington, USA, April 1992.

[97] P. Veríssimo and L. Rodrigues. Group orientation: a paradigm for modern distributed systems. In *Proceedings of the 5th ACM SIGOPS European Workshop*, Mont Saint-Michel, France, September 1992.

[98] P. Veríssimo, L. Rodrigues, and A. Casimiro. Cesiumspray: a precise and accurate global time service for large-scale systems. *Journal of Real-Time Systems*, 12(3):243–294, 1997.

[99] Paulo Veríssimo and C. Almeida. Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models. *Bullettin of the Technical Committee on Operating Systems and Application Environments (TCOS)*, 7(4):35–39, 1995.

[100] Paulo Veríssimo and António Casimiro. The timely computing base. DI/FCUL TR 99–2, Department of Computer Science, University of Lisbon, May 1999.

[101] Paulo Veríssimo, António Casimiro, and Christof Fetzer. The timely computing base: Timely actions in the presence of uncertain timeliness. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 533–542, New York City, USA, June 2000. IEEE Computer Society Press.

[102] Paulo Veríssimo, António Casimiro, and Christof Fetzer. The timely computing base: Timely actions in the presence of uncertain timeliness. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 533–542, New York City, USA, June 2000. IEEE Computer Society Press.

[103] Paulo Veríssimo, Nuno Ferreira Neves, and Miguel Correia. The middleware architecture of MAFTIA: A blueprint. In *Proceedings of the IEEE Third Information Survivability Workshop (ISW-2000)*, Boston, Massachusetts, USA, October 2000.

[104] Paulo Veríssimo and Michel Raynal. Time, clocks and temporal order. In S. Krakowiak and S. Shrivastava, editors, *Advances in Distributed Systems*, volume 1752 of *Lecture Notes in Computer Science*, chapter 1. Springer-Verlang, 2000.

[105] Paulo Verssimo, Lus Rodrigues, and Antnio Casimiro. Cesiumspray: a precise and accurate global time service for large-scale systems. *Journal of Real-Time Systems*, 12(3):243–294, May 1997.

[106] J. Xu, B. Randell, A. Romanovsky, C. Rubira, R. J. Stroud, and Z. Wu. Fault tolerance in concurrent object-oriented software through coordinated error recovery. In *25th International Symposium on Fault-Tolerant Computing*, pages 499–508, 1995.

[107] J. Xu, B. Randell, A. Romanovsky, R. J. Stroud, A. F. Zorzo, E.Canver, and F. v. Henke. Rigorous development of a safety-critical system based on coordinated atomic actions. In *29th International Symposium on Fault-Tolerant Computing*, pages 68–75, 1999.

[108] J. Xu, A. Romanovsky, and B. Randell. Concurrent exception handling and resolution in distributed object systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(11):1019–1032, 2000.