Journal of Computer Security 00 (2005) 1–36
IOS Press

Specifying and using intrusion masking models to process distributed operations

Meng Yu [a], Peng Liu [b,*] and Wanyu Zang [b]
[a] *Department of Computer Science, Monmouth University, West Long branch, NJ 07764, USA*
[b] *School of Information Sciences and Technology, The Pennsylvania State University, University Park, PA 16802, USA*

It is important for critical applications to provide critical services without any integrity or availability degradation in the presence of intrusions. This requirement can be satisfied by intrusion masking techniques under some situations. Compared with intrusion tolerance techniques, where some integrity or availability degradations are usually caused, intrusion masking techniques use substantial replications to avoid such degradations. Existing intrusion masking techniques, such as the state machine approach, can effectively mask intrusions when processing requests from a client using a server replica group, but they are fairly limited in processing a (multi-stage) distributed operation across multiple server replica groups. As more and more applications (e.g., supply chain management, distributed banking) need to process distributed operations in an intrusion-masking fashion, it is in urgent need to overcome the limitations of existing intrusion masking techniques. In this paper, we specify and compose two intrusion-masking models for inter-replica-group distributed computing. Using these two models, a variety of applications can mask (numerous kinds of) intrusions. Our intrusion masking models overcome the limitations of existing intrusion masking techniques. The survivability of our intrusion-masking models is quantitatively analyzed. A simple yet practical implementation method of our intrusion-masking models is proposed and applied to build two intrusion-masking two-phase-commit (2PC) protocols, and the corresponding efficiency is analyzed. The two intrusion-masking 2PC protocols and the analysis results show that the proposed intrusion-masking models have good utility, practicality, and survivability. Finally, the composition methodology developed in this paper can also be used to develop other intrusion-masking distributed computing models.

Keywords: Intrusion masking, survivable systems, distributed systems, security

1. Introduction

1.1. Background

Computer systems are designed to satisfy specific specifications. When behaviors of a system do not meet its specification, we say that the system is *faulty*. Although we would try our best to build reliable, bug free systems, unfortunately, from the point of view of software engineering, vulnerabilities cannot be totally removed from software systems. Therefore, hackers can find these vulnerabilities if they stick to

*Corresponding author. Tel.: 814-863-0641; Fax: 814-865-6426; E-mail: pliu@ist.psu.edu

looking for them. Consequently, successful attacks (i.e., intrusions) always happen and may cause serious damage to the system.

In the literature of building a reliable system, two types of faults have been studied for a few decades: *fail-stop faults* and *Byzantine faults* [20]. If a system stops doing anything during a failure, the fault that causes the failure is a fail-stop fault. If a system demonstrates arbitrary behaviors during a failure, the corresponding fault is a Byzantine fault. When an attacker successfully breaks into a system, he may simply crash the system or manipulate the system to do anything he wants. In the first case, the system demonstrates a fail-stop fault. In the latter case, the system demonstrates a Byzantine fault. Therefore, attackers' behaviors, or effects of intrusions, can usually be modeled as Byzantine failures [20]. In this paper, we focus on intrusions (on distributed systems) that can be modeled as Byzantine failures, and the term of *intrusions* and the term of Byzantine faults are interchangeably used. Note that we will not address fail-stop faults in the rest of the paper.

In the development of secure and reliable systems, considerable effort is devoted to making a system robust in the face of a variety of Byzantine faults. When a failure happens in a system, although the system may deviate from its specification for some time, if the system can sooner of later meet its specification as long as there are no further faulty actions, the system is *fault tolerant* (in terms of the fault that causes the failure). If no faulty actions caused by a fault can violate the specification of the system, we say that the system can *mask* the fault [6]. Note that a system that can mask a fault must be able to tolerate the fault.

Similarly, *intrusion tolerance* techniques, e.g., intrusion tolerant architectures [25,40], intrusion detection [21], damage containment [23], isolation [24], and attack recovery [5,48], try to identify intrusions, locate damage, reconfigure and recover a system automatically so that intrusions can be tolerated. However, in intrusion tolerant systems, attacks and intrusions will typically cause degradation in service integrity or availability. By contrast, *intrusion masking* techniques try to operate a system through attacks without service integrity or availability degradation. In other words, an intrusion masking system is able to deliver correct and sustained services to users despite intrusions that cause some portions of the system to behave in an arbitrary or malicious manner.

In the literature, intrusion masking is typically achieved by replications [6,7,13, 34,45]. These techniques replicate either services or data. In case some replicas are compromised, these techniques guarantee that when enough number of correct replicas are there, the client can still get correct responses.

In particular, most of these techniques adopt the *state machine approach* [39], which implements an intrusion-masking server (modeled as a state machine) by replicating that server (i.e., both services and data) and running a (server) replica on each of the nodes in a distributed system. In the state machine approach, given the same sequence of requests (from probably a set of clients) to each replica, a group of non-faulty replicas which start consistent (i.e., having the same state) will remain consistent (after the sequence of requests are processed). Hence, when a group of

server replicas is serving a set of clients, if the requests of the clients can be delivered to the replicas in such a way that the same sequence of requests will always be received by each replica, then if the group has $2t + 1$ replicas, it can mask t intruded replicas, since each client can use majority voting to identify both the correct and the malicious responses.

Ensuring that the same sequence of requests will be delivered to each replica is, however, fairly difficult, due to the complexities of the networking environment and the fact that any node or (communication) link in a distributed system could be faulty or vulnerable. For one example, if we let a replica be the (designated) *sender* that transmits the clients' messages (or requests) to the other replicas, then if the sender is faulty, then the group of replicas can receive inconsistent requests. On the other hand, even if the sender is not faulty, communication failures can still cause replicas to receive inconsistent or differently-ordered requests. For another example, if we let each client directly send its requests to each replica, then even if nothing is faulty, two replicas could receive two requests from two clients, respectively, in different orders, due to such reasons as delay and competition. According to [39], two requirements need to be satisfied to achieve this goal: (a) *Consistency*. Every nonfaulty server replica receives every request. (b) *Total Order*. Every nonfaulty replica processes the requests it receives in the same relative order. Developing the protocols that can satisfy these two requirements has raised a tremendous amount of interests, and fortunately as a result, a family of reliable totally ordered group communication services [1,4,17,29,35,38] which can satisfy the two requirements are developed. And these protocols (or services) have naturally become a key component of a typical implementation of the state machine approach.

However, although how to use a replica group to serve a client in an intrusion masking way is well studied in the literature, researchers have paid very little attention to interactions among replica groups, which are fairly different from the interactions between a single host (i.e., a client) and a replica group. In particular, the differences can be better illustrated using the scenario shown in Fig. 1, where two inter-connected (server) replica groups (i.e., A and B) are processing a specific request of a client. And the major differences between the two types of interactions are as follows. (a) In client-to-group interactions, only one host can send requests and receive responses, while in group-to-group interactions, any replica of Group A can send requests and receive responses when Group A wants Group B to process a request. (b) The message passing schemes between Group A and Group B are much more complicated than those between the client and Group A. Note that replica groups A and B consist of replicas of two different servers. Note also that using multiple servers to process a single request is not unusual in many real world applications. For example, in the banking industry, processing a request (e.g., a fund transfer request) may involve a set of interactions among two or more banks in many cases.

These differences lead to several interesting research issues, which are the focus of this paper. For example, when a n-member group A sends a request to another

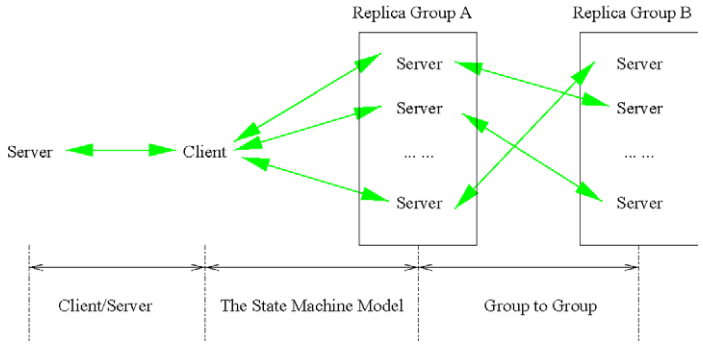


Fig. 1. Three types of interactions in distributed systems.

m-member group B, which message passing schemes should the two replica groups use to interact with each other? Should each member in A send a message to one member in B, or to all members in B? If each member in B can receive messages from only one specific member in A, how to know if the messages are faulty (or malicious)? Moreover, we hope that distributed operations across a set of replica groups can be executed in an intrusion-masking manner, thus which intrusion masking model should the replica groups use? To which degree can these approaches mask intrusions? How efficient are these approaches?

Although some of these issues are addressed in [39], where the idea of tolerating faulty clients using a group of replicated clients is discussed, the discussion is very preliminary and most of the questions we raised above are left unanswered. In addition, the fact that more and more real world critical servers have already been replicated to replica groups for reliability, availability, and survivability also indicates the need for practical, cost-effective intrusion-masking inter-replica-group distributed computing platforms. For example, when most banks replicate their database servers, every distributed transaction across a couple of branches within the same bank, or across a couple of banks, is actually executed on top of a set of replica groups. Transactions across banks (or branches) are treated as atomic operations and such atomicity is typically provided by the two-phase commit protocol (2PC) or the three-phase commit protocol (3PC). However, neither 2PC nor 3PC is designed for running on top of replica groups. Do they need to be changed for correct executions on top of replica groups? If so, how should they be adapted? In this paper, we not only address general intrusion-masking models for inter-replica-group distributed computing, but also address some specific application of these models such as intrusion-masking 2PC and 3PC protocols.

1.2. Our contributions

Our main contributions are as follows. First, we formally specify a group-to-group communication service with the I/O automaton model proposed by Lynch and Tut-

tle [27]. The group-to-group communication service handles message passing between replica groups. Compared with existing group communication services, ours has the following merits.

- *Accept consistent messages from a non-faulty group*. A replica group is *non-faulty* if the number of faulty members is no more than one third of the group. The group-to-group communication service accepts consistent inputs from non-faulty groups even if some of the members of the sending group are faulty. It delivers totally ordered messages to replica groups.
- *Fault tolerant*. When the receiving group is a non-faulty group, all non-faulty members of the receiving group can get consistent messages if the sending group is a non-faulty replica group.
- *Abstraction*. Our specification does not describe all the potentially useful properties of an application built on top of the communication service. Instead, it includes only the properties that are needed for applications to do ordered-communications. Nevertheless, our preliminary analysis results suggest that our specification is also useful for satisfying some other needs of applications.

Second, we build specifications of some building blocks of the intrusion masking models we are going to develop. In particular, the building blocks are a replica, a multicast channel, a totally ordered group-communication service, and a voting machine. The specification of a replica specifies a replicated server with both replicated data and replicated service. The replica processes requests according to their arriving order. The multicast channel specifies a general multicast service that is not fault tolerant and provided by normal network services. The totally ordered group-communication service specifies a reliable group-communication service that accepts inputs from a single host. The voting machine accepts majority of inputs as its input to resist faults. All the building blocks are easy to implement and obtain. They can be combined to construct complex distributed systems. Their combinations have not been well studied in the literature, and many features or properties of such combinations are not clearly understood.

Third, based on the state-machine model, we construct two composite intrusion-masking models by integrating the group-to-group communication service and the set of building blocks. Using these two generic intrusion masking models, a variety of distributed applications can mask intrusions. Fourth, we analyze the survivability and efficiency of both of the two composite models. According to the analysis results, our approaches can effectively mask intrusions and are practical. Finally, we apply the two intrusion-masking distributed computing models to develop two intrusion masking 2PC protocols across replica groups.

1.3. Organization of the paper

The rest of this paper is organized as follows. We start from introducing some important properties and assumptions about distributed systems in Section 2. In

Section 3, we specify and compose two intrusion-masking models for inter-replica-group distributed computing. In particular, first, we discuss two possible message passing schemes among replica groups: symmetric message passing and asymmetric message passing. Second, we construct a simple state machine that accepts messages from a replica group and discuss its limitations. Third, we specify a group-to-group communication service and a set of building blocks. Fourth, we compose two intrusion masking models using the group-to-group communication service and the set of building blocks. Finally, we extend BFT [7] to implement the proposed models and demonstrate the feasibility of our techniques, and develop two intrusion masking 2PC protocols across replica groups to demonstrate the utility of our techniques.

The survivability and efficiency of our approaches are analyzed in Section 4 and Section 5. We address the related work in Section 6. Section 7 concludes the paper with a few suggestions on future work.

2. Preliminaries

In this section, we introduce some concepts and describe the assumptions used in this paper.

2.1. System properties

A system is *not faulty* if and only if it satisfies its specifications. The specification of a system describes a variety of properties of the system. In this paper, we are mainly concerned with safety, liveness, and survivability properties of systems.

The *safety* property of a system specifies the functions that a non-faulty system should perform. In this paper, we specify the safety properties of an intrusion masking system using the I/O automaton model proposed by Lynch and Tuttle [27]. The model and its proof methods can be found in Chapter 8 of [28]. Since the two intrusion-masking models presented in this paper are both composite models, and complicated and subtle relationships among the set of building blocks (i.e., component models) are usually involved in these models, we need such formal methods as the I/O automaton model to remove the ambiguity in specifying and proving the correctness of these composite intrusion masking models.

The *liveness* property requires that a request to a system can always get a response. The intrusion masking solutions proposed in this paper are built on top of consensus protocols (among replicas) to ensure the safety or security, but it has been proved that in an asynchronous system consensus cannot be implemented [10]. Therefore, our approaches rely on synchrony to provide liveness.

Another important property of a system is survivability, which has been defined in several different ways in the literature [8,14,18,22]. However, since these definitions either are qualitative thus they cannot be used for quantitative analysis of survivability, or are defined for a specific environment, e.g., survivability of networks, none of them can be directly applied here.

In this paper, we define *survivability* of a system as follows.

Definition 1. The survivability of a given system under a given condition is the probability that the system can provide liveness and safety.

For example, a crashed system that does not respond to any request has survivability 0 and a normally functioned web server has survivability 1. A web server group with 1 normally functioned server and 1 faulty server that never responds has survivability 0.5 if any of them does not forward requests to the other and does not answer requests from the other. In other words, a client has probability 0.5 to obtain the service correctly from the server group.

2.2. Assumptions

2.2.1. Intrusions

In this paper, we assume a hacked system may demonstrate arbitrary behaviors that are usually modeled as Byzantine faults [20].

Note that although our intrusion model can handle every type of insider or outsider attacks on a server (replica) and many types of attacks on the client (e.g., a hacked client can send inconsistent requests to a server replica group), some intrusions cannot be handled by our approaches. For example, malicious transactions submitted by authorized but malicious clients to corrupt server data cannot be handled by our approaches. They can be handled by transaction level intrusion tolerance techniques [5,48], which are complementary to our approaches.

2.2.2. Replica group

We call a set of replicated servers a *replica group*. Each replicated server is a *replica*. Replicas in the same replica group have consistent data, services, and initial states. In our model, a *service* is a sequence of *operations*.

A replica group consists of several replicas that have the same initial status (or state) $\mathscr{S}$ and operation (or services) set $\mathscr{O}$. We use $\mathscr{G}_1, \mathscr{G}_2, \ldots, \mathscr{G}_n$ to denote replica groups. We use $r_{i,1}, r_{i,2}, \ldots, r_{i,k}$ to represent replicas belonging to ith replica group $\mathscr{G}_i$, where $\mathscr{G}_i$ is called a k-size-group if $|\mathscr{G}_i| = k$. For simplicity, any non-replicated host is treated as a 1-size-group in the rest of this paper.

For a distributed system composed of n replica groups, i.e., $\mathscr{G}_1, \mathscr{G}_2, \ldots, \mathscr{G}_n$, we assume any two groups will not overlap each other, that is, if $i \neq k$, then $\mathscr{G}_i \cap \mathscr{G}_k = \phi$. In addition, we use $\mathscr{R} = \mathscr{G}_1 \cup \mathscr{G}_2 \cup \ldots \cup \mathscr{G}_n$ to denote the universal set of replicas, and we use $\mathscr{U} = \{\mathscr{G}_1, \mathscr{G}_2, \ldots, \mathscr{G}_n\}$ to denote the universal set of groups.

The safety properties for a replica are specified in Fig. 2.

In the specification, a replica is an I/O automaton denoted R. The automaton is composed of a set of *input* operations, a set of *output* operations, and a set of *internal* operations. A *state* of the automaton is defined by a specific set of state *variables*. In particular, variable $ready$ denotes the set of requests sent to R; variable $last$ demotes the total count of requests; variable $result$ denotes the set of results generated by processing the requests in $ready$; variable $faulty - replica$ indicates whether R is faulty. The initial values of the set of state variables are called the *initial state* of R.

In R, a state *transition* can be caused by an input operation, an output operation, or an internal process. For one example, when a replica group $\mathscr{G}$ asks R to perform operation o, the corresponding input operation REQUEST$(o)_{\mathscr{G}}$ will add a new request into $ready$ and increase $last$ by 1; the corresponding internal operation EXECUTE$(o, i, \mathscr{G})$ will perform o (when o is the next one to handle) and generate an answer a (which is part of $result$); then the corresponding output operation REPLY$(a)_r$ will send the answer a to each replica r of $\mathscr{G}$. In our specification, the *effects* (Eff for short) of each operation type are specified. In addition, the specification of an operation may include a set of *preconditions* (Pre for short), which must be satisfied before the operation is performed. In our specification, a replica strictly executes and replies requests according to their arrival order. The function $\mathscr{F}(o, \mathscr{S})$ maps the current state $\mathscr{S}$ and the requested operation o to a new state and an answer $a \in \mathscr{A}$. For any two replicas in the same replica group, if we can guarantee that they always receive the same requests in the same order, they will always result in the same state $\mathscr{S}$, as the state machine approach does.

In our model, a replica *accepts* an request from a group if it receives consistent requests from all members of the group. Otherwise, the replica will not accept the request and the corresponding operation will not be performed. In our specification,

Signature:

Input:	REQUEST$(o)_{\mathscr{G}}, o \in \mathscr{O}, \mathscr{G} \in \mathscr{U}$
	REPLICA-FAILURE
Internal:	EXECUTE$(o, i, \mathscr{G}), o \in \mathscr{O}, i \in \mathbb{N}, \mathscr{G} \in \mathscr{U}$
Output:	REPLY$(a)_r, a \in \mathscr{A}, r \in \mathscr{R}$
	SREPLY$(a)_r, a \in \mathscr{A}, r \in \mathscr{R}$

State:

$ready \subseteq \mathscr{O} \times \mathbb{N} \times \mathscr{U}$, init ϕ
$result \subseteq \mathscr{A} \times \mathbb{N} \times \mathscr{U}$, init ϕ
$last \in \mathbb{N}$, init 0
faulty-replica $\in Bool$, init $false$
$\mathscr{S}$, initial value depends on the replica group

Transitions(if *faulty-replica* $= false$**):**

input REQUEST$(o)_{\mathscr{G}}$
Eff: $ready \leftarrow ready \cup \{\langle o, last, \mathscr{G}\rangle\}$
$last \leftarrow last + 1$

input REPLICA-FAILURE
Eff: *faulty-replica* $= true$

output REPLY$(a)_r$
Pre: $\langle a, i, \mathscr{G}\rangle \in result$, where
$i = \min(\{i' \mid \langle a, i', \mathscr{G}\rangle \in result\})$
Eff: $result \leftarrow (result - \{\langle a, i, \mathscr{G}\rangle\})$
$\cup\{\langle a, i, \mathscr{G} - \{r\}\rangle\}$

internal EXECUTE$(o, i, \mathscr{G})$
Pre: $\langle o, i, \mathscr{G}\rangle \in ready$, where
$i = \min(\{i' \mid \langle o, i', \mathscr{G}\rangle \in ready\})$
Eff: $(a, \mathscr{S}) \leftarrow \mathscr{F}(o, \mathscr{S})$
$result \leftarrow result \cup \langle a, i, \mathscr{G}\rangle$
$ready \leftarrow ready - \{\langle o, i, \mathscr{G}\rangle\}$

output SREPLY$(a)_r$
Pre: $\langle a, i, \mathscr{G}\rangle \in result$, where $r \in \mathscr{G}$,
$i = \min(\{i' \mid \langle a, i', \mathscr{G}\rangle \in result\})$
Eff: $result \leftarrow result - \{\langle a, i, \mathscr{G}\rangle\}$

Fig. 2. R.

R offers two different kinds of outputs. REPLY provides replies to all members of the group that sends requests. SREPLY provides replies to only one member of the group who sends requests. Behaviors of a faulty replica are left unspecified to simulate Byzantine faults.

Since replicas may be faulty, replicas in the same group may have different states. When we get results or receive responses from a replica group, we need to know how many consistent messages are enough to guarantee that the results are correct or valid. A size-n-group usually can resist no more than $f = \lfloor \frac{n-1}{3} \rfloor$ faulty members that have Byzantine faults [7,20]. Accordingly, first, we define a specific majority measurement for replica groups, that is, for a group G composed of n members, we define $majority(\mathscr{G}) = 2f + 1$. (Note that our majority definition is very different from the normal majority definition where a sub-group of size $n/2+1$ is a majority.) Second, we call any size-n-group that has more than $\lfloor \frac{n-1}{3} \rfloor$ faulty members a *faulty group*.

2.2.3. Communication environment

For simplicity, we assume any two hosts in a distributed system have a "connection" protected by cryptography in such a way that no message between any two hosts can be modified or forged. Nevertheless, messages may be delayed without boundary. Hence, *asynchronous* communications are assumed.

We define a multicast service which is specified in Fig. 3. MC does not guarantee delivery orders while it eventually delivers all messages. In this paper, our discussions about the relationships between the number of faulty hosts and the survivability of a system are based on the MC service. Since persistent communication faults can be handled as processor faults, we do not try to conclude results related to the number of faulty connections. Although using unicast to implement MC can easily ensure that no message between any two hosts can be modified or forged, ensuring

Signature:

Input: MC-SEND$(m, \mathscr{G})_r, m \in \mathscr{M}, \mathscr{G} \in \mathscr{U}, r \in \mathscr{R}$
Output: MC-RECEIVE$(m)_r, m \in \mathscr{M}, r \in \mathscr{R}$

State:

$wire \subseteq \mathscr{R} \times \mathscr{M} \times \mathscr{U}$, init ϕ

Transitions:

input MC-SEND$(m, \mathscr{G})_r$
Eff: $wire \leftarrow wire \cup \{\langle r, m, \mathscr{G}\rangle\}$

output MC-RECEIVE$(m)_r$
Pre: $\langle r', m, \mathscr{G}\rangle \in wire, r \in \mathscr{G}$
Eff: $wire \leftarrow wire - \{\langle r', m, \mathscr{G}\rangle\}$
$\cup\{\langle r', m, \mathscr{G} - \{r\}\rangle\}$

Fig. 3. MC.

this property when broadcast is used to implement MC is not that straightforward since a compromised replica knowing the encryption key could be used to cause damage to the other replicas. Nevertheless, this problem can be solved in several ways (e.g., using digital signatures) and such details are out of the scope. Finally, in this paper we do not consider connectivity and network partitions either.

3. Composing intrusion masking inter-replica-group computing models

In this section, first, we discuss two possible message passing schemes among replica groups. Second, we construct a simple state machine that accepts messages from a replica group and discuss its limitations. Third, we specify a group-to-group communication service and a set of building blocks. Fourth, we compose two intrusion masking models for inter-replica-group distributed computing using the group-to-group communication service and the set of building blocks. Finally, we present an implementation of the proposed intrusion masking models to demonstrate the feasibility of our techniques, and develop two intrusion masking 2PC protocols across replica groups to demonstrate the utility of our techniques.

3.1. Message passing schemes

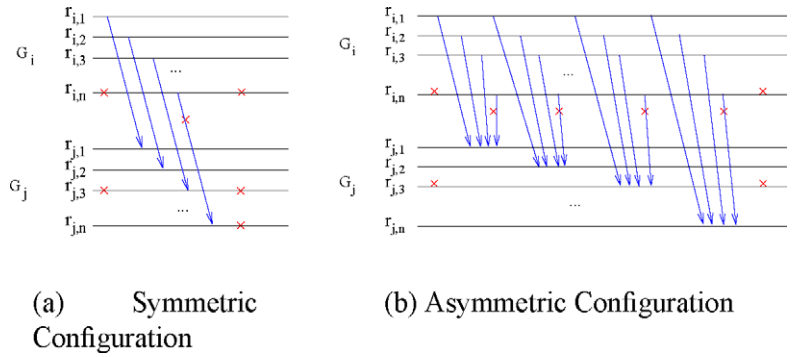
When replica groups want to interact with each other, numerous inter-group message passing schemes are possible. Here, we focus on two frequently used schemes.

3.1.1. Symmetric configuration

If the size of all replica groups are the same, one method of message passing is *symmetric sending*. For example, when a size-m-group $\mathscr{G}_i$ wants to send a request r to a size-n-group $\mathscr{G}_j$ and m is equal to n, we can pass messages in an one-to-one style. Symmetric sending can be constructed by a *working group*. In our model, a working group is a group of replicas that satisfy the following conditions:

1. A member of a working group only interacts with members in the same working group.
2. A replica belongs to only one working group.
3. Any two replicas in the same replica group belong to different working groups.

Formally, suppose there are m size-n-groups $\mathscr{G}_1, \mathscr{G}_2, \ldots, \mathscr{G}_m$ in the system. For all $\mathscr{G}_i, 1 \leqslant i \leqslant m$, $|\mathscr{G}_i| = n$. We construct *working groups set* $\mathscr{W} = \{\{r_{1,i}, r_{2,j}, \ldots, r_{m,k}\} \mid r_{1,i} \in \mathscr{G}_1, r_{2,j} \in \mathscr{G}_2, \ldots, r_{m,k} \in \mathscr{G}_m\}$, where $|\mathscr{W}| = n$, $\forall w \in \mathscr{W} : |w| = m$, $\forall w_i, w_j \in \mathscr{W} : i \neq j \Rightarrow w_i \cap w_j = \phi$, $\forall r, r' \in w, w \in \mathscr{W}$, $r \neq r' : r \in \mathscr{G} \Rightarrow r' \notin \mathscr{G}$. Each element w of $\mathscr{W}$ is a working group. For all $\mathscr{G}_i, 1 \leqslant i \leqslant m$, we say $\mathscr{W}$ *contains* $\mathscr{G}_i$. Every replica in a size-n-group is assigned to a working group. Every two elements of a working group come from different replica groups.



(a) Symmetric Configuration (b) Asymmetric Configuration

Fig. 4. Examples of symmetric and asymmetric configurations.

In Fig. 4(a), $\mathscr{G}_i = \{r_{i,1}, r_{i,2}, \ldots, r_{i,n}\}$ and $\mathscr{G}_j = \{r_{j,1}, r_{j,2}, \ldots, r_{j,n}\}$ are two size-n-groups. We can construct $\mathscr{W} = \{\{r_{i,1}, r_{j,1}\}, \{r_{i,2}, r_{j,2}\}, \ldots, \{r_{i,n}, r_{j,n}\}\}$. Each element of $\mathscr{W}$, $\{r_{i,l}, r_{j,l}\}, 1 \leqslant l \leqslant n$, is a working group. Please note that the construction of set $\mathscr{W}$ is not unique according to our definition.

After a working group set $\mathscr{W}$ that contains $\mathscr{G}_i$ and $\mathscr{G}_j$ is constructed, for all $r_{i,k} \in \mathscr{G}_i$, $r_{i,k}$ sends request $\langle req \rangle_{\sigma_{r_{i,k}}}$ to $r_{j,l}$ if and only if $\exists w \in \mathscr{W} : r_{i,k}, r_{j,l} \in w$ and $r_{j,l} \in \mathscr{G}$. Here, $\sigma_{r_{i,k}}$ means that req should be signed by $r_{i,k}$. Note that when there are no faulty replicas, the req parts in all these (request) messages from $\mathscr{G}_i$ to $\mathscr{G}_j$ should be the same because they are the same request sent by different replicas. Figure 4(a) shows an example of symmetric sending. In a distributed system, if all replica groups interact with each other using symmetric sending, we denote such a configuration by *symmetric configuration* or G-SC.

3.1.2. Asymmetric configuration

Another message passing pattern is *asymmetric sending*. Messages are passed in a many-to-many style. When a size-m-group $\mathscr{G}_i$ wants to send a request r to a size-n-group $\mathscr{G}_j$, no matter m is equal to n or not, for all $r_{i,k} \in \mathscr{G}_i$, $r_{i,k}$ sends request $\langle req \rangle_{\sigma_{r_{i,k}}}$ to every replica in $\mathscr{G}_j$. Then totally mn messages are sent. Each replica in $\mathscr{G}_j$ will receive m messages signed by different replicas in $\mathscr{G}_i$. An example is shown in Fig. 4(b). In the example, $r_{i,2}$ interacts with all the replicas in $\mathscr{G}_j$, such as $r_{j,1}, r_{j,2}, \ldots, r_{j,n}$. In a distributed system, if all replica groups interact with each other using asymmetric sending, we denote such a configuration as *asymmetric configuration* or G-AC.

It should be noticed that symmetric configuration and asymmetric configuration have different efficiency and capability of fault tolerance. In Fig. 4, replicas marked with a cross are faulty replicas and messages marked with a cross are faulty messages. Symmetric sending has less communication cost but less capability of fault tolerance than those of asymmetric sending. A fault (e.g., $r_{i,n}$) in the sending replica group (e.g., $\mathscr{G}_i$) will be propagated to the receiving group (e.g., $r_{j,n}$) in symmetric sending, while asymmetric sending may be able to resist such fault propagations.

3.2. An extension to the state machine approach and limitations

An intuitive idea to build intrusion-masking inter-replica-group distributed systems is to extend the classic state-machine approach to handle a replica group as a "single" client. The standard state-machine approach considers only single hosts as clients. Our "intuitive" extension has two parts: (1) we specify a new, totally-ordered communication service, denoted TO, to accept requests from group clients. TO is specified in Fig. 5(a). (2) We use TO and R to compose an extended state machine model that can handle group clients. The composite model is shown in Fig. 5(b).

In TO-R, two automatons are *combined* as follows: a directed edge from one automaton A_1 to another automaton A_2 always starts with an output action (or operation) of A_1 and ends with an input action of A_2. This means that the output action always triggers the input action. For example, in Fig. 5(b), a TO-RECEIVE operation of TO always triggers a REQUEST operation of R.

Signature:

Input: TO-SEND$(m, \mathcal{G})_{\mathcal{G}'}, m \in \mathcal{M}, \mathcal{G}, \mathcal{G}' \in \mathcal{U}$
REPLICA-FAILURE$_r, r \in \mathcal{R}$
Output: TO-RECEIVE$(m)_r, m \in \mathcal{M}, r \in \mathcal{R}$

State:

for each $r \in \mathcal{R}$, *faulty-replica*$_r \in Bool$, init $false$
n-faulty$(\mathcal{G}) \equiv |\{r \mid$ *faulty-replica*$_r = true, r \in \mathcal{G}\}|$
$wire \subseteq \mathcal{U} \times \mathcal{M} \times \mathbb{N} \times \mathcal{U}$, init ϕ
$last \in \mathbb{N}$, init 0

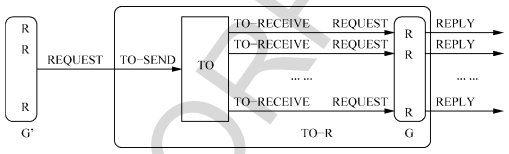
Transitions (if *n-faulty*$(\mathcal{G}) \leqslant \lfloor \frac{|\mathcal{G}|-1}{3} \rfloor$**):**

input TO-SEND$(m, \mathcal{G})_{\mathcal{G}'}$
Eff: $wire \leftarrow wire \cup \{\langle \mathcal{G}', m, last + 1, \mathcal{G} \rangle\}$
$last \leftarrow last + 1$

input REPLICA-FAILURE$_r$
Eff: *faulty-replica*$_r = true$

output TO-RECEIVE$(m)_r$
Pre: $\langle \mathcal{G}', m, i, \mathcal{G} \rangle \in wire$, where $r \in \mathcal{G}$,
$i = \min(\{i' \mid \langle \mathcal{G}', m, i', \mathcal{G} \rangle \in wire\})$,
faulty-replica $= false$
Eff: $wire \leftarrow wire - \{\langle \mathcal{G}', m, i, \mathcal{G} \rangle\}$
$\cup \{\langle \mathcal{G}', m, i, \mathcal{G} - \{r\} \rangle\}$

(a)



(b)

Fig. 5. Compose a state machine with totally ordered service. (a) TO. (b) An intuitive extension of the state machine approach to handle group clients.

Since TO-R is a composite state machine automaton, it has its own input and output operations, which are simply a subset of the unions of the input and output operations of all of its component automatons, respectively. In particular, the REPLY output operation of TO-R is *offered by* the REPLY output operation of R, and the REQUEST input operation of TO-R is *offered by* the TO-SEND input operation of TO.

TO has the following properties:

1. TO *is totally ordered.* From one perspective, when a client replica group is sending two requests (which have no causal relation between them) to a server replica group, these two requests may reach two server replicas in different orders. This property ensures that every server replica will always receive (and process) the requests from the client group in the same order. From another perspective, when two client replica groups A and B are sending requests to a server replica group, a request from A and a request from B may reach two server replicas in different orders. This property ensures that every server replica will always receive (and process) requests from multiple client groups in the same order.
2. TO *is Byzantine fault tolerant.* The capability of fault tolerance of TO is characterized by the number of faulty members in the receiving group because TO is usually implemented inside the receiving group as a software component of the receiving group. TO can mask (or tolerate) up to $\lfloor \frac{|\mathcal{G}|-1}{3} \rfloor$ malicious members of the receiving group. This number adopts the result of some existing techniques for resisting Byzantine faults, such as BFT [7]. The case that $n\text{-}faulty(\mathcal{G}) > \lfloor \frac{|\mathcal{G}|-1}{3} \rfloor$ is left unspecified to simulate arbitrary faults.
3. TO *accepts* a request from a replica group only if TO receives consistent requests from all replicas of the requesting group.

Although TO-R can process requests from a replica group, TO-R is not a practical intrusion masking computing model since it cannot mask (or tolerate) any faulty member of the requesting group. Even a non-faulty group that has only one faulty replica cannot be accepted by the TO-R machine.

3.3. A Practical solution

In this section, we overcome the limitations of TO-R by presenting two practical, composite, intrusion-masking models for inter-replica-group distributed computing. Compared with the intuitive extension of the state machine approach, these two models has two significant advantages:

- Both models can mask a specific number of faulty replicas of the client group (though to some extent).
- Both models allow a set of server-replica-groups to collaboratively process a distributed operation issued by the client. Note that the standard state machine approach cannot handle interactions among server-replica-groups.

However, before we present the composite models, we need to first specify several building blocks.

3.3.1. Totally Ordered Group-to-Group Communications

Inter-replica-group intrusion-masking computing needs appropriate group-to-group communication services. In this section, we specify a totally ordered group communications service that has multiple inputs. The specification is shown in Fig. 6. To our best knowledge, this is the first specification of such group communication services.

MTO has two types of inputs. MTO-SSEND$(r, m, \mathcal{G})_{r', \mathcal{G}'}$ is for symmetric configurations and MTO-ASEND$(r, m, \mathcal{G})_{\mathcal{G}'}$ is for asymmetric configurations. When a

Signature:

Input:	MTO-ASEND$(r, m, \mathcal{G})_{\mathcal{G}'}, m \in \mathcal{M}, \mathcal{G}, \mathcal{G}' \in \mathcal{U}, r \in \mathcal{G}$
	MTO-SSEND$(r, m, \mathcal{G})_{r', \mathcal{G}'}, m \in \mathcal{M}, \mathcal{G}, \mathcal{G}' \in \mathcal{U}, r \in \mathcal{G}, r' \in \mathcal{G}'$
	REPLICA-FAILURE$_r, r \in \mathcal{R}$
Internal:	A-RECEIVED$(\mathcal{G}', m, \mathcal{G}), m \in \mathcal{M}, \mathcal{G}, \mathcal{G}' \in \mathcal{U}$
	S-RECEIVED$(\mathcal{G}', m, \mathcal{G}), m \in \mathcal{M}, \mathcal{G}, \mathcal{G}' \in \mathcal{U}$
Output:	MTO-RECEIVE$(m)_r, m \in \mathcal{M}, r \in \mathcal{R}$

State:

for each $r \in \mathcal{R}$, *faulty-replica*$_r \in Bool$, init $false$ $\quad last \in \mathbb{N}$, init 0
n-faulty$(\mathcal{G}) \equiv |\{r \mid$ *faulty-replica*$_r = true, r \in \mathcal{G}\}|$
$pending \subseteq \mathcal{R} \times \mathcal{M} \times \mathcal{U}$,
$\cup \mathcal{R} \times \mathcal{U} \times \mathcal{R} \times \mathcal{M} \times \mathcal{U}$, init ϕ
$wire \subseteq \mathcal{U} \times \mathcal{M} \times \mathbb{N} \times \mathcal{U}$, init ϕ

Transitions (if *n-faulty*$(\mathcal{G}) \leqslant \lfloor \frac{|\mathcal{G}|-1}{3} \rfloor$**):**

input MTO-ASEND$(r, m, \mathcal{G})_{\mathcal{G}'}$
Eff: $pending \leftarrow pending \cup \{\langle \mathcal{G}', r, m, \mathcal{G} \rangle\}$

input MTO-SSEND$(r, m, \mathcal{G})_{r', \mathcal{G}'}$
Eff: $pending \leftarrow pending \cup \{\langle r', \mathcal{G}', r, m, \mathcal{G} \rangle\}$

input REPLICA-FAILURE$_r$
Eff: *faulty-replica*$_r = true$

internal A-RECEIVED$(\mathcal{G}', m, \mathcal{G})$
Pre: $\mathcal{S} = \{\langle \mathcal{G}', r, m, \mathcal{G} \rangle \mid \langle \mathcal{G}', r, m, \mathcal{G} \rangle \in pending$, *faulty-replica*$_r = false\}$,
$|\mathcal{S}| \geqslant majority(\mathcal{G})$,
Eff: $wire \leftarrow wire \cup \{\langle \mathcal{G}', m, last + 1, \mathcal{G} \rangle\}$
$pending = pending - \mathcal{S}$
$last \leftarrow last + 1$

internal S-RECEIVED$(G', m, \mathcal{G})$
Pre: $|\mathcal{G}| = |\mathcal{G}'|$,
$\mathcal{S} = \{\langle r', \mathcal{G}', r, m, \mathcal{G} \rangle \mid \langle r', \mathcal{G}', r, m, \mathcal{G} \rangle \in pending$, *faulty-replica*$_r = false\}$,
$|\mathcal{S}| \geqslant majority(\mathcal{G})$, $\forall \langle r_1, \mathcal{G}', r_2, m, \mathcal{G} \rangle$, $\langle r_3, \mathcal{G}', r_4, m, \mathcal{G} \rangle \in \mathcal{S} : seq(r_1) = seq(r_2) \Rightarrow seq(r_3) = seq(r_4)$, $seq(r_3) = seq(r_4) \Rightarrow seq(r_1) = seq(r_2)$
Eff: $wire \leftarrow wire \cup \{\langle \mathcal{G}', m, last + 1, \mathcal{G} \rangle\}$
$pending = pending - \mathcal{S}$
$last \leftarrow last + 1$

output MTO-RECEIVE$(m)_r$
Pre: $\langle \mathcal{G}', m, i, \mathcal{G} \rangle \in wire$, where $r \in \mathcal{G}$,
$i = \min(\{i' \mid \langle \mathcal{G}', m, i', \mathcal{G} \rangle \in wire\})$,
Eff: $wire \leftarrow wire - \{\langle \mathcal{G}', m, i, \mathcal{G} \rangle\}$
$\cup \{\langle \mathcal{G}', m, i, \mathcal{G} - \{r\} \rangle\}$

Fig. 6. MTO.

requesting group $\mathscr{G}'$ is sending a request to a receiving group $\mathscr{G}$, $|\mathscr{G}|$ input operations will be performed by MTO under either symmetric sending or asymmetric sending. Under symmetric sending, each MTO-SSEND involves a pair of replicas from $\mathscr{G}$ and $\mathscr{G}'$, respectively; and any replica of $\mathscr{G}$ will not be part of two MTO-SSEND operations. Under asymmetric sending, each MTO-SSEND involves a replica of $\mathscr{G}$ and all the replicas of $\mathscr{G}'$. Compared with TO, MTO not only keeps the *total-order* property and the *Byzantine fault tolerance* property, but also can tolerate up to $\lfloor \frac{|\mathscr{G}'|-1}{3} \rfloor$ malicious members of the requesting group. For example, internal operation A-RECEIVED will allow a replica r of $\mathscr{G}$ to accept a request from $\mathscr{G}'$ as soon as r receives $majority(\mathscr{G})$ consistent copies of the request from $majority(\mathscr{G})$ replicas of $\mathscr{G}'$. Moreover, it should be noticed that MTO will not receive (inconsistent) requests from faulty replica groups for correctness. Finally, MTO can be implemented in many ways, and we will present an implementation shortly.

3.3.2. Voting machine

We design a voting machine, shown in Fig. 7, to handle messages sent from non-faulty replica groups that have faulty replicas. If V receives consistent messages from more than $majority(\mathscr{G}')$ replicas of the requesting group $\mathscr{G}'$, V will accept the message and believe that the message is from a non-faulty group.

3.3.3. Composite models

Now we are ready to present the two intrusion masking models, namely C_1 and C_2. C_1 is built for symmetric configurations, while C_2 is built for asymmetric configurations. C_1 is composed of MTO and R. C_1 is shown in Fig. 8(a). Here, output operation $\text{REPLY}(a)_{r',r}$ indicates that r' gets reply, i.e., answer a, from r.

Signature:

Input: V-IN$(m)_{r,\mathscr{G}}, \mathscr{G} \in \mathscr{U}, m \in \mathscr{M}, r \in \mathscr{R}$
Output: V-OUT$(m)_{\mathscr{G}}, \mathscr{G} \in \mathscr{U}, m \in \mathscr{M}$

State:

$in \subseteq \mathscr{U} \times \mathscr{R} \times \mathscr{M}$, init ϕ

Transitions:

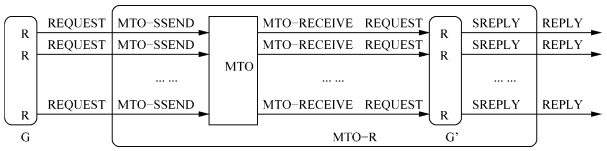
input V-IN$(m)_{r,\mathscr{G}}$
Eff: $in \leftarrow in \cup \{\langle \mathscr{G}, r, m \rangle\}$

Output V-OUT$(m)_{\mathscr{G}}$
Pre: $\mathscr{S} = \{\langle \mathscr{G}, r, m \rangle \mid \langle \mathscr{G}, r, m \rangle \in in, r \in \mathscr{G}\}$,
$|S| \geqslant majority(\mathscr{G})$
Eff: $in \leftarrow in - \mathscr{S}$

Fig. 7. V.

In other words, when one replica group $\mathscr{G}'$ sends a (valid) request to another replica group $\mathscr{G}$, $\mathscr{G}$ will perform exactly $|\mathscr{G}|$ output operations where each output operation will allow a replica r of $\mathscr{G}$ sends an instance of the reply to a specific replica r' of $\mathscr{G}'$.

Theorem 1. *C_1 implements* MTO-R.



(a)

Signature:

Input:	REQUEST$(r, o, \mathscr{G})_{r', \mathscr{G}'}, o \in \mathscr{O}, \mathscr{G}, \mathscr{G}' \in \mathscr{U}, r \in \mathscr{G}, r' \in \mathscr{G}'$
	REPLICA-FAILURE$_r, r \in \mathscr{R}$
Internal:	RECEIVED$(\mathscr{G}', o, \mathscr{G}), o \in \mathscr{O}, \mathscr{G}, \mathscr{G}' \in \mathscr{U}$
	EXECUTE$(\mathscr{G}', o, i, \mathscr{G}), o \in \mathscr{O}, i \in \mathbb{N}, \mathscr{G}, \mathscr{G}' \in \mathscr{U}$
Output:	REPLY$(a)_{r', r}, a \in \mathscr{A}, r, r' \in \mathscr{R}$

State:

$pending \subseteq \mathscr{R} \times \mathscr{U} \times \mathscr{R} \times \mathscr{O} \times \mathscr{U}$, init ϕ
$ready \subseteq \mathscr{U} \times \mathscr{O} \times \mathbb{N} \times \mathscr{U}$, init ϕ
$result \subseteq \mathscr{U} \times \mathscr{A} \times \mathbb{N} \times \mathscr{U}$, init ϕ
$last \in \mathbb{N}$, init 0
n-faulty$(\mathscr{G}) \equiv |\{r \mid$ *faulty-replica*$_r = true, r \in \mathscr{G}\}|$
faulty-replica$_r \in Bool$, init $false$

Transitions(if *n-faulty*$(\mathscr{G}) \leqslant \lfloor \frac{|\mathscr{G}|-1}{3} \rfloor$**):**

input REQUEST$(r, o, \mathscr{G})_{r', \mathscr{G}'}$
Eff: $pending \leftarrow pending \cup \{\langle r', \mathscr{G}', r, o, \mathscr{G}\rangle\}$
input REPLICA-FAILURE$_c$
Eff: *faulty-replica*$_r = true$
internal RECEIVED$(G', o, \mathscr{G})$
Pre: $|\mathscr{G}| = |\mathscr{G}'|$,
$\mathscr{S} = \{\langle r', \mathscr{G}', r, o, \mathscr{G}\rangle \mid \langle r', \mathscr{G}', r, o, \mathscr{G}\rangle \in pending$, *faulty-replica*$_r = false\}$,
$|\mathscr{S}| \geqslant majority(\mathscr{G}), \forall \langle r_1, \mathscr{G}', r_2, o, \mathscr{G}\rangle, \langle r_3, \mathscr{G}', r_4, o, \mathscr{G}\rangle \in \mathscr{S} : seq(r_1) = seq(r_2) \Rightarrow seq(r_3) = seq(r_4),\ seq(r_3) = seq(r_4) \Rightarrow seq(r_1) = seq(r_2)$
Eff: $ready \leftarrow ready \cup \{\langle \mathscr{G}', o, last + 1, \mathscr{G}\rangle\}$
$pending = pending - \mathscr{S}$
$last \leftarrow last + 1$

internal EXECUTE$(\mathscr{G}', o, i, \mathscr{G})$
Pre: $\langle \mathscr{G}', o, i, \mathscr{G}\rangle \in ready$, where
$i = \min(\{i' \mid \langle \mathscr{G}', o, i', \mathscr{G}\rangle \in ready\})$
Eff: $result \leftarrow result \cup \langle \mathscr{G}', \mathscr{F}(o), i, \mathscr{G}\rangle$
$ready \leftarrow ready - \{\langle \mathscr{G}', o, i, \mathscr{G}\rangle\}$
output REPLY$(a)_{r', r}$
Pre: $\langle \mathscr{G}', a, i, \mathscr{G}, \rangle \in result, r \in \mathscr{G}$,
$r' \in \mathscr{G}'$, *faulty-replica*$_r = false$,
$i = \min(\{i' \mid \langle \mathscr{G}', a, i', \mathscr{G}\rangle \in result\})$
Eff: $result \leftarrow result - \{\langle \mathscr{G}', a, i, \mathscr{G}\rangle\}$
$\cup \{\langle \mathscr{G}' - \{r'\}, a, i, \mathscr{G} - \{r\}\rangle\}$

(b)

Fig. 8. Composite model for G-SC. (a) Composite model C_1. (b) MTO-R.

MTO-R is specified in Fig. 8. In this following, we prove this theorem by showing that there exists a *simulation relation (or mapping)* (denoted $\mathscr{F}()$) between C_1 and MTO-R that satisfies the following requirements.

1. $\mathscr{F}()$ maps every initial state of C_1 to an initial state of MTO-R.
2. $\mathscr{F}()$ maps every reachable state of C_1 to a reachable state of MTO-R.
3. For each step (or state transition) of C_1, denoted (s, π, s'), there is an execution fragment α of MTO-R that goes from $\mathscr{F}(s)$ to $\mathscr{F}(s')$, such that $trace(\alpha) = trace(\pi)$.

We construct $\mathscr{F}(s)$ as follows:

- We let $\mathscr{O} \subseteq \mathscr{M}$, that is, a requested operation $o \in \mathscr{O}$ could be transmitted as a message.
- If a state variable (e.g., $result$) belongs to a specific component automaton of C_1 (e.g., a specific replica r), we annotate the state variable with a subscript that denotes the component, e.g., $result_r$.
- Let s be a state of C_1. $\mathscr{F}()$ maps s to t, a state of MTO-R as follows.
  - $t.last = s.last$
  - $t.pending = s.pending$
  - $t.ready = s.wire$
  - $\forall \langle \mathscr{G}', a, i, \mathscr{G} \rangle \in t.result$: $\langle a, i, \mathscr{G} \rangle \in s.result_r$ and $r \in \mathscr{G}'$.
  - *t.replica-failure*$_r$ = *s.replica-failure*$_r$

We prove the correctness of $\mathscr{F}()$ using two lemmas which are specified as follows.

Lemma 1. *If s is an initial state of C_1, then $\mathscr{F}(s)$ is an initial state of* MTO-R.

Lemma 2. *Let s be a reachable state of C_1, $\mathscr{F}(s)$ a reachable state of* MTO-R*, and (s, π, s') a step of C_1. Then there is an execution fragment α of* MTO-R *that goes from $\mathscr{F}(s)$ to $\mathscr{F}(s')$, such that $trace(\alpha) = trace(\pi)$.*

Proof sketch. Consider all the actions (or operations) that automatons MTO and R take.

- $\pi =$ MTO-ASEND. This action is disabled in C_1.
- $\pi =$ MTO-SSEND. This action offers the input for requests. π is enabled and it adds inputted message to $pending$. Let $\alpha =$ REQUEST and α does the same thing in MTO-R.
- $\pi =$ REPLICA-FAILURE. Let $\alpha =$ REPLICA-FAILURE. Since $\mathscr{F}(s) = s = t$, so $(\mathscr{F}(s), \alpha, \mathscr{F}(s'))$ is a step of MTO-R.
- $\pi =$ A-RECEIVE. This action is not an action of MTO-R. Since MTO-ASEND is disabled in C_1, according to the codes of MTO-ASEND there are no elements in $pending$ like $\langle \mathscr{G}', r, m, \mathscr{G} \rangle$. Hence, the precondition of action π cannot be satisfied. So, π is disabled.

- $\pi =$ S-RECEIVE. π is not an action of MTO-R. Let $\alpha =$ RECEIVE. π and α have the same precondition and effects. We map $\mathscr{F}(s) = s = t$. So, $trace(\pi) = trace(\alpha) = \lambda$, namely, no external actions.
- $\pi =$ MTO-RECEIVE. π has the same precondition as EXECUTE in MTO-R. Let $\alpha =$ EXECUTE. In our composition, we consider MTO-RECEIVE$(m)_{r_i}$ is the ith output of MTO and the output is connected with the ith replica R. π triggers an action REQUEST of R followed by an action EXECUTE$_{r_i}$ of the ith R. So, $\pi =$ MTO-RECEIVE $\cdot$ REQUEST$_r \cdot$ EXECUTE$_r$. They are not actions of MTO-R. According to our codes, after taking these actions, π and α lead to the same state since they take the same actions on $ready(wire)$ and $result$.
- $\pi =$ REQUEST. See that $\pi =$ MTO-RECEIVE.
- $\pi =$ EXECUTE. See that $\pi =$ MTO-RECEIVE.
- $\pi =$ REPLY. This action is disabled in C_1.
- $\pi =$ SREPLY. Let $\alpha =$ REPLY. According to the definition of $\mathscr{F}$, if $\exists\langle a, i, \mathscr{G}\rangle \in s.result_{r_j}$, then $\exists\langle \mathscr{G}' a, i, \mathscr{G}\rangle \in t.result$ and $r \in \mathscr{G}'$. So, the jth R in C_1 and the jth output of MTO-R are enabled. After π and α, the jth result is removed from both $s.result$ and $t.result$. The result cannot be gotten twice in either C_1 or MTO-R. So, getting it again is disabled. □

From Lemma 1 and Lemma 2 we have:

Theorem 2. *$\mathscr{F}$ is a simulation relation.*

Therefore, Theorem 1 has proved.
We conclude the following theorem directly from the composition.

Theorem 3. MTO-R *can survive no more than* $\lfloor \frac{n-1}{3} \rfloor$ *faulty replicas.*

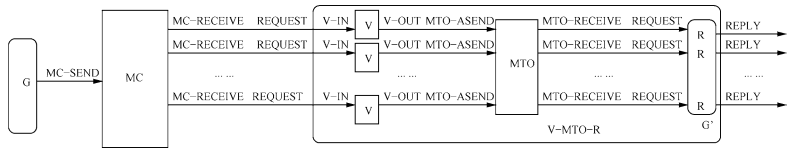
For the asymmetric configuration we compose C_2 with V, MTO and R to accept asymmetric sending inputs, which is specified in Fig. 9(a). V-MTO-R accepts inputs from a non-faulty group even if the group has some faulty replicas. The voting machine votes for inputs and ignores faulty inputs. Then, V-MTO-R votes for its own inputs to prevent malicious groups from cheating it by inconsistent requests.

Theorem 4. *C_2 implements* V-MTO-R.

Proof sketch. This theorem can be proved in a way quite similar to proving Theorem 1. □

From the composition, we have the following theorem.

Theorem 5. V-MTO-R *can survive no more than* $\lfloor \frac{n-1}{3} \rfloor$ *faulty replicas.*



(a)

Signature:

Input:	REQUEST$(r, o, \mathscr{G})_{r',\mathscr{G}'}, o \in \mathscr{O}, \mathscr{G}, \mathscr{G}' \in \mathscr{U}, r \in \mathscr{G}, r' \in \mathscr{G}'$
	REPLICA-FAILURE$_r, r \in \mathscr{R}$
Internal:	RECEIVED$(\mathscr{G}', o, \mathscr{G}), o \in \mathscr{O}, \mathscr{G}, \mathscr{G}' \in \mathscr{U}$
	EXECUTE$(\mathscr{G}', o, i, \mathscr{G}), o \in \mathscr{O}, i \in \mathbb{N}, \mathscr{G}, \mathscr{G}' \in \mathscr{U}$
Output:	REPLY$(a)_{\mathscr{G}',r}, a \in \mathscr{A}, \mathscr{G}' \in \mathscr{U}, r \in \mathscr{R}$

State:

$pending \subseteq \mathscr{R} \times \mathscr{U} \times \mathscr{R} \times \mathscr{O} \times \mathscr{U}$, init ϕ
$ready \subseteq \mathscr{U} \times \mathscr{O} \times \mathbb{N} \times \mathscr{U}$, init ϕ
$result \subseteq \mathscr{U} \times \mathscr{A} \times \mathbb{N} \times \mathscr{U}$, init ϕ
$last \in \mathbb{N}$, init 0

n-faulty$(\mathscr{G}) \equiv |\{r \mid$ *faulty-replica*$_r = true, r \in \mathscr{G}\}|$
faulty-replica$_r \in Bool$, init $false$

Transitions(if *n-faulty*$(\mathscr{G}) \leqslant \lfloor \frac{|\mathscr{G}|-1}{3} \rfloor$**):**

input REQUEST$(r, o, \mathscr{G})_{r',\mathscr{G}'}$
Eff: $pending \leftarrow pending \cup \{\langle r', \mathscr{G}', r, o, \mathscr{G}\rangle\}$
input REPLICA-FAILURE$_c$
Eff: *faulty-replica*$_r = true$
internal RECEIVED$(\mathscr{G}', r, o, \mathscr{G})$
Pre: $\mathscr{S} = \{\langle r', \mathscr{G}', r, o, \mathscr{G}\rangle \mid \langle r', \mathscr{G}', r, o, \mathscr{G}\rangle \in pending$, *faulty-replica*$_r = false\}$, $|\mathscr{S}| \geqslant majority(\mathscr{G}')$
Eff: $pending = pending - \mathscr{S} \cup \{\langle \mathscr{G}', r, o, \mathscr{G}\rangle\}$
internal RECEIVED$(\mathscr{G}', o, \mathscr{G})$
Pre: $\mathscr{S} = \{\langle \mathscr{G}', r, o, \mathscr{G}\rangle \mid \langle \mathscr{G}', r, o, \mathscr{G}\rangle \in pending\}$, $|\mathscr{S}| \geqslant majority(\mathscr{G})$
Eff: $ready \leftarrow ready \cup \{\langle \mathscr{G}', o, last + 1, \mathscr{G}\rangle\}$
$pending = pending - \mathscr{S}$
$last \leftarrow last + 1$

internal EXECUTE$(\mathscr{G}', o, i, \mathscr{G})$
Pre: $\langle \mathscr{G}', o, i, \mathscr{G}\rangle \in ready$, where $i = \min(\{i' \mid \langle \mathscr{G}', o, i', \mathscr{G}\rangle \in ready\})$
Eff: $result \leftarrow result \cup \langle \mathscr{G}', \mathscr{F}(o), i, \mathscr{G}\rangle$
$ready \leftarrow ready - \{\langle \mathscr{G}', o, i, \mathscr{G}\rangle\}$
output REPLY$(a)_{\mathscr{G}',r}$
Pre: $\langle \mathscr{G}', a, i, \mathscr{G}, \rangle \in result, r \in \mathscr{G}$, *faulty-replica*$_r = false$, $i = \min(\{i' \mid \langle \mathscr{G}', a, i', \mathscr{G}\rangle \in result\})$
Eff: $result \leftarrow result - \{\langle \mathscr{G}', a, i, \mathscr{G}\rangle\} \cup \{\langle \mathscr{G}', a, i, \mathscr{G} - \{r\}\rangle\}$

(b)

Fig. 9. Composite model for G-AC. (a) Composite model C_2. (b) V-MTO-R.

Now we can carry out a distributed operation in two possible ways. (1) Under the G-SC configuration, every replica group is constructed as a MTO-R machine. Every replica group sends requests and obtains replies from other groups using symmetric (message) sending. (2) Under the G-AC configuration, every replica group is

constructed as a V-MTO-R machine. Every replica group sends requests and obtains replies from other groups using asymmetric sending. Note that any distributed operation (e.g., 2PC protocol operations) can be processed in these two ways, and our intrusion masking computing models are generic models.

3.4. Case study: building intrusion masking 2PC protocols

In this section, first, we show how BFT [7], a popular and practical Byzantine fault tolerance protocol, can be used to implement both C_1 and C_2. Second, we show how such implementations can be applied to build practical intrusion masking 2PC protocols.

3.4.1. Extending BFT to BFT-S

BFT is a practical Byzantine fault tolerance protocol that enables a server replica group to handle requests from single-host clients. BFT ensures that when there are no more than $\lfloor \frac{n-1}{3} \rfloor$ faulty replicas in the size-n-server-group, (a) all the non-faulty replicas of the server group will process the requests from multiple clients in the same order; (b) every request they process will be consistent; and (c) the clients will receive correct, consistent replies. Moreover, BFT achieves practical performance.

Based on BFT, we introduce a new Byzantine fault tolerant protocol named BFT-S to meet the requirements of our communication schemes among replica groups. BFT-S is specified as follows and is shown in Fig. 10. Here, we assume the client (group) interacts with a size-$3k + 1$-(server)-group. k is the maximum number of Byzantine-faulty replicas in the (server) group that can be masked or tolerated. In our architecture, either a single node or a replica group could be the client. Note that here we focus on the differences of BFT-S from BFT. Note also that our techniques are not limited to BFT and can be applied to other similar protocols based on the state machine approach.

Phase request and phase reply. When the client is a size-$3k + 1$-group and the system is under symmetric configuration, the client symmetrically sends the $\langle \text{SYMMETRIC-REQUEST}, o, t, c \rangle_{\sigma_c}$ message to the size-$3k + 1$-server-group. In the request, t is the time stamp and o is the operation (requested to be performed). c is a replica of the client group. At the end of BFT-S, each replica i of the size-$3k + 1$-server-group symmetrically sends reply $\langle \text{SYMMETRIC-REPLY}, v, t, c, i, r \rangle_{\sigma_i}$ to the client, where r is the result of request t and v is the view number.

On the other hand, when the system is under asymmetric configuration (the client can be either a single node or a replica group), or the client is a single node and the system is under symmetric configuration, the client asymmetrically sends the $\langle \text{ASYMMETRIC-REQUEST}, o, t, c \rangle_{\sigma_c}$ message to the size-$3k + 1$-server-group. c is the client or a replica of the client. If the client is a size-$3j + 1$-group and the system is under asymmetric configuration, each replica in the size-$3k + 1$-server-group should not start to process a request

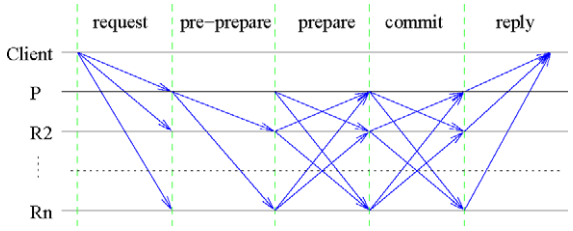


Fig. 10. An normal BFT-S procedure.

until $j + 1$ matching instances of the request are received. When BFT-S ends, the reply $\langle \text{ASYMMETRIC-REPLY}, v, t, c, i, r \rangle_{\sigma_i}$ will be asymmetrically sent to the client. The client will wait for a *weak certificate* before accept the reply as the result. The weak certificate is composed of $k + 1$ matching instances (of the reply) that have a valid signature of a server replica, and the same v, t and r. Note that the term "weak certificate" comes from BFT [7] since it only requires $k + 1$ consistent messages rather than $2k + 1$, the usual threshold.

Phase pre-prepare. The *primary* assigns a sequence number n to the request and multicasts the request to other server replicas in the pre-prepare phase. Each replica will accept the request if and only if the request from the client matches the request from the primary. Otherwise, it will simply discard the request. Note that in general any server replica can be the primary. Note also that BFT [7] uses a specific "view change protocol" to handle corrupted primary nodes.

The prepare phase and the commit phase are the same as BFT. They can be simply described as follows. In the prepare phase and the commit phase, each server replica broadcasts the signed request to others and collects $2k + 1$ matching requests as a certificate to authorize itself to continue the following operations. For more information about BFT please refer to [7]. A normal BFT-S procedure with single client is shown in Fig. 10.

BFT-S improves BFT in three aspects. First, a replica group can be a client of BFT-S. Second, in BFT-S all requests from the same replica group can be processed in parallel. This property can greatly reduce the communications cost of BFT-based 2PC protocols. Third, clients do not need to know "who is the primary of a replica group?" We have the following theorem directly from BFT.

Theorem 6. *BFT-S can tolerate no more than* $\lfloor \frac{n-1}{3} \rfloor$ *faulty members of either a size-*n*-server-group or a size-*n*-client-group.*

3.4.2. BFT-S-based intrusion masking 2PC protocols

A distributed operation has different part of itself running on different hosts simultaneously. To ensure correct execution of distributed operations, we need to ensure

the "all-or-nothing" property. The 2PC (or 3PC) protocol guarantees that a distributed operation will either be successfully committed or be aborted without any effect on the system. According to the 2PC protocol, when a distributed operation is executed, the *coordinator* asks all the participants if they are ready to commit the distributed operation. If every participant answer "ready", the coordinator will order all the participants to commit the operation. Otherwise, the distributed operation will be totally aborted. The 3PC protocol is similar to the 2PC protocol.

Two intrusion masking atomic commit protocols (BFT-ACP) are introduced in this section. Our BFT-ACP protocols can be combined with either 2PC or 3PC protocols, and are named as BFT-2PC and BFT-3PC, respectively. We first discuss BFT-2PC under Symmetry Configuration (BFT-2PC-SC), then discuss BFT-2PC under Asymmetry Configuration (BFT-2PC-AC).

In our model, a client is asking a set of servers to process a distributed transaction. For intrusion masking, each of the servers are replicated to form a replica group. The client may be a replica group too. During the execution of the distributed transaction, the replica group that receives the request from the client is called the *coordinator group* and the other groups are called *participant groups*.

After accepting a distributed transaction from the client, the coordinator group carries out three phase of BFT-S to achieve an agreement, namely PRE-PREPARE, PREPARE and COMMIT. After that, the request will be logged in the stable storage for further recovery in case of malicious transactions. Then, BFT-2PC-SC or BFT-2PC-AC will be carried out according to the system configuration.

The BFT-2PC-SC protocol is described as follows.

Phase one: request to prepare. The coordinator group symmetrically sends $\langle$SYMMETRIC-REQUEST, REQUEST-TO-PREPARE, $t, c\rangle_{\sigma_c}$ to all the participant groups. Each participant group starts a BFT-S for the request to prepare.

Note that t (the time stamp) in each of the request messages should be the same. Because t is used to guarantee the semantics of "exactly once", so it will not be confusing even if two different requests have the same time stamp.

Phase two: voting on commit. The participant groups symmetrically send the reply $\langle$SYMMETRIC-REPLY, v, t, c, i, PREPARED$\rangle_{\sigma_i}$ or $\langle$SYMMETRIC-REPLY, v, t, c, i, NO$\rangle_{\sigma_i}$ to the coordinator group in terms of the transaction can be prepared or not.

Each replica in the coordinator group makes a decision for "commit" if all the replies received from the participant groups are PREPARED. Otherwise, it decides "abort". In both cases, all replicas of the coordinator group should achieve an agreement on their decisions to ensure consistency, which is implemented by multicasting the decisions of each replica among the group and collecting $2f + 1$ matching decisions if the coordinator group is a size-$3f + 1$-group.

Next, the coordinator group notifies all the participant groups to "commit" or "abort" by symmetrically sending the request $\langle$SYMMETRIC-REQUEST,

COMMIT, $t, c\rangle_{\sigma_c}$ or $\langle$SYMMETRIC-REQUEST, ABORT, $t, c\rangle_{\sigma_c}$ to all the participant groups. Each participant group then starts another BFT-S procedure for the notice. Finally, each participant group symmetrically sends the reply $\langle$SYMMETRIC-REPLY, $v, t, c, i,$ DONE$\rangle_{\sigma_i}$ to the coordinator group.

After the coordinator group receives the DONE replay, the coordinator group sends replies to the client.

The BFT-2PC-AC protocol can be described in exactly the same way as BFT-2PC-SC except replacing all symmetric sending with asymmetric sending and replacing all SYMMETRIC-REQUEST, SYMMETRIC-REPLY with ASYMMETRIC-REQUEST and ASYMMETRIC-REPLY, respectively.

4. Survivability

In this section, we evaluate the intrusion-masking capability of the two intrusion-masking distributed computing models we proposed in the previous section. Our evaluation method is to quantify the survivability (we defined in Section 2) achieved by these two intrusion-masking models. In our specification, a working group w is *correct* if and only if every replica in w is not faulty and the communications among them are not faulty. Otherwise, the working group is a *faulty* working group because the operations among the replicas of the working group cannot obtain correct results.

On one hand, the survivability of G-SC can be characterized by the following theorem.

Theorem 7. *Given a distributed system composed of a set of size-n-replica-groups, if the distributed system is under G-SC configuration* (*stated by C_1*)*, the system can survive no more than $\lfloor \frac{n-1}{3} \rfloor$ faulty working groups when carrying out a distributed operation.*

Proof sketch. In G-SC, each replica in a replica group interacts with replicas of other groups through a specific working group. If the replica belongs to a faulty working group, it will not get a correct result when carrying out a distributed operation. When the number of faulty working groups is no more than $\lfloor \frac{n-1}{3} \rfloor$, no replica groups will have more than $\lfloor \frac{n-1}{3} \rfloor$ faulty members. According to Theorem 3, any replica group can survive such failure. So, the distributed operation will be correct.

When the number of faulty working groups is more than $\lfloor \frac{n-1}{3} \rfloor$, there exists at least one communication step where the receiver will receive more than $\lfloor \frac{n-1}{3} \rfloor$ faulty messages. According to Theorem 3, the receiving group cannot resist such failure. □

Please note in Theorem 7 the faults are quantified by the number of faulty working groups instead of the number of faulty replicas in each replica group. This is because failures can be propagated across replicas within a working group. In particular, the

following example shows why the number of faulty replicas in each replica group should not be used.

Example 1. Assume two replica groups are collaborating on processing a distributed operation. Assume one group is $\{r_{i,1}, r_{i,2}, r_{i,3}, r_{i,4}\}$, and the other group is $\{r_{j,1}, r_{j,2}, r_{j,3}, r_{j,4}\}$. Assume four working groups are formed: the first replicas of each group form the first working group, and so forth. Assume only two replicas are faulty; they are $r_{i,1}$ and $r_{j,3}$. It is clear that every replica group is non-faulty. However, this distributed system cannot survive these two faulty working groups, namely working groups 1 and 3, since (a) when $r_{j,1}$ receives a malicious request from $r_{i,1}$, the fault on $r_{i,1}$ will be "propagated" to $r_{j,1}$, that is, the internal operations of $r_{j,1}$ based on the malicious request will usually be distorted; (b) as a result, replica group j cannot reach an agreement on the reply that should be sent to group i because half of the replicas of group j cannot correctly function. Moreover, from the perspective of BFT-S, replica group j cannot earn the certificate needed to commit.

Based on Example 1, we can get the following corollary.

Corollary 1. *Given a distributed system composed of m size-n-replica-groups, if the distributed system is computing under G-SC configuration* (*stated by C_1*)*, then*

- *The distributed system can mask* (*or survive*) *up to $k = \lfloor \frac{n-1}{3} \rfloor$ malicious replicas, no matter where these faulty replicas stay.*
- *When g, the total number of faulty* (*or malicious*) *replicas, is larger than k, but less than or equal to mk, i.e., $k < g \leqslant mk$, if the probability that a replica is faulty is identical throughout the whole system, the survivability P of the system can be described with Eq.* (1)*. G-SC cannot survive more than mk faulty replicas.*

$$P = \frac{\binom{n}{k}\binom{mk}{g}}{\binom{mn}{g}}. \tag{1}$$

Proof sketch. Let $k = \lfloor \frac{n-1}{3} \rfloor$. When there are k faulty replicas in the system, consider the worst case, these replicas locate in different working groups then there are totally $\lfloor \frac{n-1}{3} \rfloor$ faulty working groups in the system. According to Theorem 7, G-SC can survive such failure.

When $k < g \leqslant mk$, according to Theorem 7 all g (faulty replicas) should stay in no more than $\lfloor \frac{n-1}{3} \rfloor$ working groups to ensure that G-SC will survive the failure. There are totally $\binom{n}{k}\binom{mk}{g}$ solutions that put g faulty replicas in no more than $\lfloor \frac{n-1}{3} \rfloor$ working groups while there are totally $\binom{mn}{g}$ solutions that put g faulty replicas in mn positions in the system. Based on the assumption, the probabilities that g faulty replicas occur at each group are equal, hence the survivability P of system is exactly the expression of Eq. (1).

When the number of faulty replicas is more than mk, there is no way to arrange these faulty replicas in less than $\lfloor \frac{n-1}{3} \rfloor$ working groups, and according to Theorem 7, G-SC cannot survive such failure. □

Remark. In Corollary 1 and Corollary 8 below, we assume the probability that a replica is faulty is identical throughout the whole system. It should be noticed that this assumption captures the scenarios where the inter-replica-group distributed computing system may achieve the maximum amount of survivability. Although when the system is poorly protected, the attacker may be able to select a set of well-selected replicas to attack, this assumption enables us to derive the theoretical upper-bound of the survivability of the proposed intrusion masking models, and it is practically supported by the emerging technologies to diversify systems (and their vulnerabilities) and to hide the memberships within a replica group.

On the other hand, the survivability of G-AC can be characterized by the following theorem.

Theorem 8. *Given a distributed system composed of a set of replica groups, G-AC (or C_2) can survive no more than $\lfloor \frac{n-1}{3} \rfloor$ faulty replicas in each size-n-group of the system.*

Proof sketch. Because asymmetric sending can resist propagation of faults, so, G-AC can function correctly if and only if every replica group in the system functions correctly. Based on Theorem 5, this theorem is correct. □

The following corollary gives a lower bound of the survivability of G-AC.

Corollary 2. *Given a distributed system composed of m size-n-replica-groups, if the distributed system is computing under G-AC configuration, and if the probability that a replica is faulty is identical throughout the whole system, then G-AC can survive at least $k = \lfloor \frac{n-1}{3} \rfloor$ faulty replicas, no matter where these faulty replicas stay. Beside this, if $k < g \leqslant mk$ (g denotes the total number of faulty replicas), the survivability P of the system can be characterized with Eq.* (2)*. G-AC cannot survive more than mk faulty replicas.*

$$P \geqslant \max\left(0, \frac{\binom{mn}{g} - m\sum_{i=k+1}^{\min(g,n)} \binom{n}{i}\binom{(m-1)n}{g-i}}{\binom{mn}{g}}\right). \tag{2}$$

Proof sketch. Let $k = \lfloor \frac{n-1}{3} \rfloor$. When there are k faulty replicas in the system, according to Theorem 8, G-AC can survive such failure because there is no such replica group that has more than $\lfloor \frac{n-1}{3} \rfloor$ faulty replicas.

When $k < g \leqslant mk$, according to Theorem 8, all g faulty replicas should stay in the "good" *locating mode*, where there is no such replica group that has more than $\lfloor \frac{n-1}{3} \rfloor$ faulty replicas, to ensure that G-AC will survive the failure.

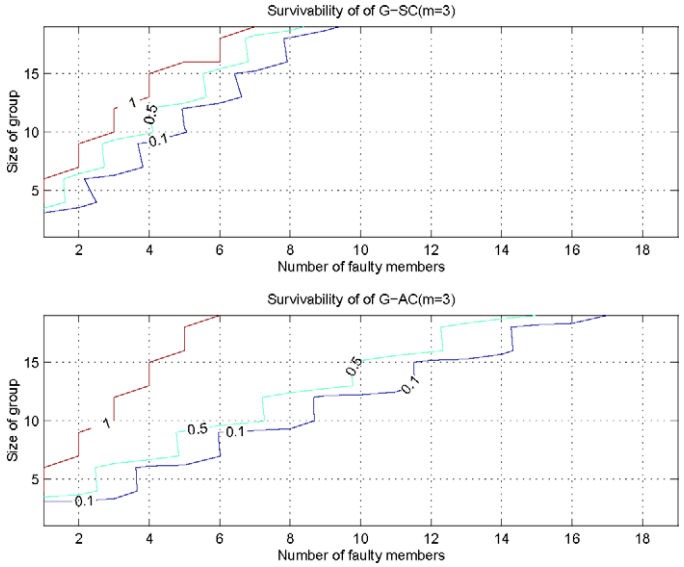


Fig. 11. The survivability of G-SC and G-AC while $m = 3$.

To calculate the probability of the "good" node is not trivial. First, we can count the "bad" modes where G-AC cannot survive. To construct a locating mode, we first select one replica group, and there are m solutions. Then we assign $\lfloor \frac{n-1}{3} \rfloor + 1 = k+1$ faulty replicas in the replica group. Finally, we assign $g - (k+1)$ faulty replicas to other replica groups. The total number of solutions for this locating configuration is $m \sum_{i=k+1}^{\min(g,n)} \binom{n}{i} \binom{(m-1)n}{g-i}$. Unfortunately, in fact the bad modes have been repeatedly counted when m is big enough.

Because there are totally $\binom{mn}{g}$ solutions that put g faulty replicas in mn positions in the system, so, the number of locating modes that G-AC can survive is more than $\binom{mn}{g} - m \sum_{i=k+1}^{\min(g,n)} \binom{n}{i} \binom{(m-1)n}{g-i}$. Based on the assumption, the probabilities that g faulty replicas occur at each group are equal, hence the survivability P of the system can be characterized by Eq. (2).

When the number of faulty replicas more than mk, there is no way to arrange these faulty replicas to ensure that there is no such replica group that has more than $\lfloor \frac{n-1}{3} \rfloor$ faulty replicas. According to Theorem 8, G-AC cannot survive such failure. □

Note that Corollary 2 gives a lower bound of survivability of G-AC. When m is small enough (e.g., $m = 2$ or $m = 3$), the corollary gives the accurate survivability.

When there are m size-n-groups in the system, we compared the expected survivability of G-SC and G-AC. The expected survivability when $m = 3$ is shown in Fig. 11. When m has different values, the subtractions of G-SC's survivability from G-SC's survivability are shown in Fig. 12. We observed that G-AC always has

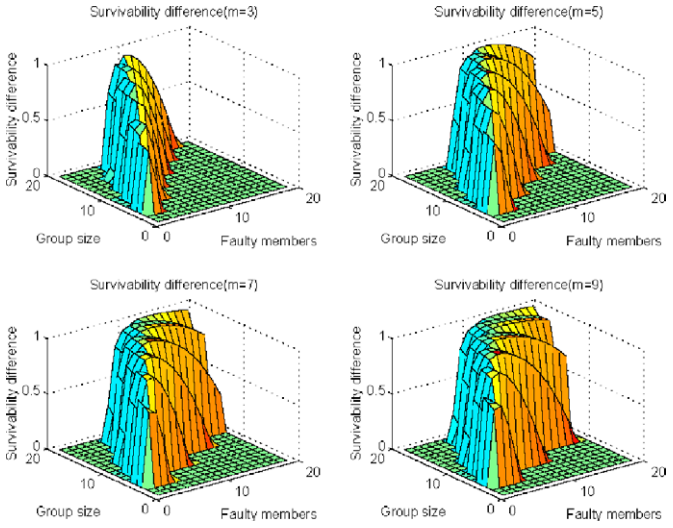


Fig. 12. The subtractions of G-SC's survivability from G-AC's survivability.

better survivability than that of G-SC. We believe this is because asymmetric sending has better capability of fault tolerance. The asymmetric structure of G-AC also contributes to its survivability.

5. Efficiency

In this section, we evaluate the efficiency of the two intrusion-masking distributed computing models we proposed in Section 3. The efficiency of the two intrusion-masking models can be characterized in several aspects. In this paper, we focus on the communication costs and the latency.

The communication costs can be quantified by the number of messages, which depends on the implementation of the communication layer. For example, multicast could be implemented by unicast or broadcast in the lower layer. The communication costs of these two implementation methods are very different. So we count the (number of) messages for both cases.

Suppose a distributed operation consists of a sequence of $M_1, C_1, M_2, C_2, \ldots, M_k, C_k$, where $M_i, 1 \leqslant i \leqslant k$, is a step of message passing and $C_i, 1 \leqslant i \leqslant k$, is a step of computing. Under G-SC, there are totally $k(f(n) + n)$ messages, where $f(n)$ is the number of messages consumed to achieve a consensus within the MTO machine. Under G-AC, there are totally $\sum_{i=1}^{k}(f(n_k) + n_k n_{k+1})$ messages.

In the rest of the paper, we investigate a more practical scenario where 2PC operations are performed across a set of replica groups under symmetric sending and

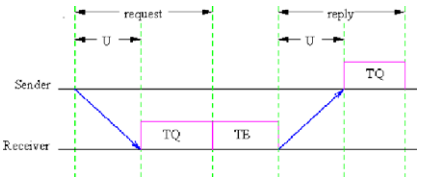


Fig. 13. Time model.

Table 1
Communication cost of BFT-2PC-SC and BFT-2PC-AC

	BFT-2PC-SC	BFT-2PC-AC	$n_1 = n_2 = \cdots = n_m$
N_{pI}	$(4m-3)n$	$n + 4\sum_{i=2}^{m} n_1 n_i$	$4(m-1)n^2 + n$
N_{pL}	$6n^2 - 3n - 3$	$2n^2 - n - 1 + 2\max(2n_i^2 - n_i - 1)$	$6n^2 - 3n - 3$
N_{bI}	$2mn + 1$	$1 + 2(m-1)n_1 + 2\sum_{i=2}^{m} n_i$	$4(m-1)n + 1$
N_{bL}	$6n + 3$	$2n + 1 + 2\max(2n_i + 1)$	$6n + 3$
N_{sig}	14	14	14
N_q	$6n + 2m - 2$	$2n + 2(n_1 - 1) + 2(2\max(n_i) - 1) + 2\sum_{i=2}^{m} n_i$	$(2m+6)n - 4$

asymmetric sending, respectively. In particular, we evaluate the efficiency of BFT-2PC-SC and BFT-2PC-AC respectively. Here we assume MTO is implemented by an adapted BFT protocol, namely BFT-S [46].

As for the latency of processing, we adopt the time model shown in Fig. 13, which is similar to the time model of BFT. In the figure, U is the multicasting time. All messages received by a receiver are queued in the buffer of the receiver. The receiver needs time TQ to process the message queue. Finally, the receiver needs time TE to execute the request.

Based on our time model, the latency of a (message passing) procedure can be calculated by Eq. (3) based on the critical path of the procedure.

$$L = \sum U + \sum TE + \sum TQ. \tag{3}$$

Because all participants can execute in parallel, so in the critical path, TE can be calculated by Eq. (4).

$$\sum TE = T_T + 2T_C + 2\max T_P + t_\sigma N_{sig}. \tag{4}$$

In the equation, T_T is the executing time of the transaction (involved in the 2PC protocol); T_C is the time of executing one BFT-S protocol by the coordinator group; T_P is the time of executing one BFT-S protocol by a participant group; T_V is the time of voting in the coordinator group; t_σ is the signature time for a single message, and N_{sig} is the total number of messages that need be signed on the critical path.

The communication costs of BFT-2PC-SC and BFT-2PC-AC are shown in Table 1. In the table, the configuration of BFT-2PC-SC is m size-n-groups under symmetric

Table 2
The parameters for latency evaluation

Symbol definitions	Default values
T_T: executing time of a transaction	1.2 second
T_C: time of one BFT-S procedure by coordinator	0.1 second
T_P: the longest time of one BFT-S by a participant	0.1 second
t_σ: signature time of a single message	0.1 second
t_I: time for transmitting a message on the Internet	0.02 second
t_L: time for transmitting a message locally	0.002 second
t_q: time to process a message in the queue	0.005 second

configuration. The configuration of BFT-2PC-AC is that the coordinator group is a size-n_1-group, the participant groups are composed of a size-n_2-group, a size-n_3-group, $\cdots$, and a size-n_m-group under asymmetric configuration. In Table 1 N_{pI} represents the number of messages transmitted on the Internet by point-to-point. N_{pL} represents the number of messages transmitted locally by point-to-point. N_{bI} represents the number of messages transmitted on the Internet by broadcasting. N_{bL} represents the number of messages transmitted locally by broadcasting.

Despite the impact of the network topology and the implementation details, for simplicity, we assume the transmitting time U is in proportion to the total number of messages N_I and N_L. We also assume that TQ is in proportion to the total length of message queue N_q on the critical path. Then we have Eq. (5).

$$\sum U + \sum TQ = t_I N_I + t_L N_L + t_q N_q. \tag{5}$$

In the equation, t_m is the unit time to transmit a message, and t_q is the unit time to process a message in the queue. So, we have the latency described with Eq. (6).

$$L = T_T + 2T_C + 2\max T_P + t_\sigma \sum N_{sig} + t_I N_I + t_L N_L + t_q N_q. \tag{6}$$

Suppose we have m size-n-groups, they can be configured in either symmetric sending or asymmetric sending. We compared their communication costs and latency. The parameters for latency evaluation are shown in Table 2.

Some default values about transactions and communications in the table are chosen based on [26]. Based on the parameters for latency evaluation, and the latency equation (i.e., Eq. (6)), we can compute the latency for a distributed transaction with either BFT-2PC-SC or BFT-2PC-AC. Because the implementation of multicast has great impact on communication costs and the latency of transactions, so we compare the latencies under two different implementations of multicast.

The latencies of the two intrusion masking 2PC protocols are shown in Fig. 14 and Fig. 15. According to the results shown in the figures, BFT-2PC-AC is not a very pragmatic scheme due to its poor latency performance, while BFT-2PC-SC demonstrates better results. Also the total number m of replica groups has great impact on

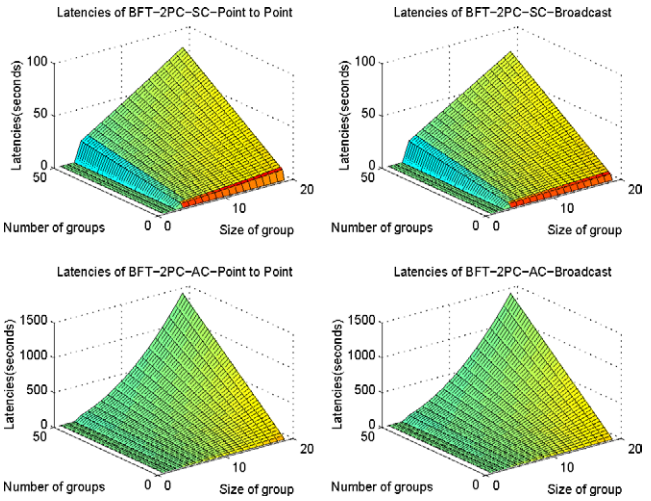


Fig. 14. Latencies under different configurations.

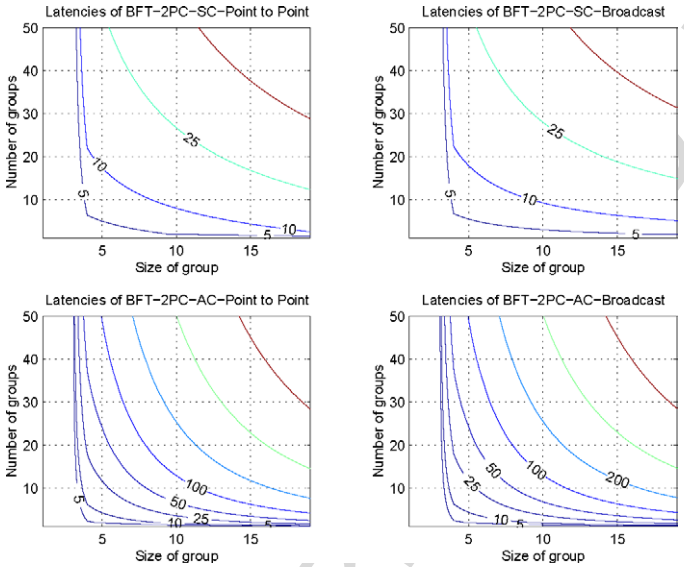


Fig. 15. Contours of latencies.

the latency of BFT-2PC-AC. Nevertheless, as shown in Fig. 15, when the size of a group is not larger than 5 and the number of groups is less than 10, the latencies of both BFT-2PC-AC and BFT-2PC-AC are shorter than 25 s. Hence, both BFT-2PC-SC and BFT-2PC-AC are practical under such settings.

So far, we ignored the impact of the network topology and the implementation details. In fact, the latency in multicast communications is not exactly proportional to the total number of messages and should actually be shorter than what we calculated in Eq. (6). So, the real efficiency of BFT-2PC-AC may be better. Still, BFT-2PC-SC has better performance than BFT-2PC-AC if we consider this optimization.

6. Related work

The work similar to ours can be classified into four categories which are discussed as follows.

6.1. Replication-based fault tolerance

Significant research efforts have been seen in search of replication algorithms that achieve both high performance and availability [2,9,15,26,37,42], in which updates are produced by a master replica then broadcasted to other replicas. A serious limitation of such systems is that the master replica may be faulty.

PASIS [11,41,43,44] is designed as a survivable storage system. It provides the stable storage we need. As we mentioned earlier, reliable storage, though an important technique, is not enough to tolerate Byzantine faults because the services which produce data may be faulty.

To handle Byzantine faults, the basic idea is to duplicate both services and data, namely full replication. The state machine approach [39] is one of such methods. Some reliable group communication services [1,4,17,29,35,38] can be valuable building blocks of the state-machine-approach. The standard state machine approach and its building blocks can handle single-host clients very well, but they are not adequate in handling replica group clients and interactions among a set of server replica groups. BFT [7] is a special type of state machine methods and is considered as a practical scheme for Byzantine fault tolerance. Unfortunately, it has no solution for carrying out operations among replica groups.

Quorum replication [3,30–32] can also have the capability of intrusion masking, but quorum replications usually restricts (the type of) operations (read and write) and needs locks to access variables. State machine approach based algorithms have no limitation on operations and could have better performance as BFT does.

6.2. Byzantine fault tolerance

Arbitrary faults or intrusions are usually modeled as Byzantine faults [20]. Byzantine agreement algorithms can resist some kinds of malicious attacks, but they are mainly for achieving consensus and not affordable in many, if not most, real world distributed environments due to their high costs.

Applying Byzantine agreement (BA) to database systems has been discussed in [12], which suggests that the only necessary application of BA is submission of transactions. Furthermore, full replication is necessary. This idea is the same as the state machine approach, but it has no solution for handling replica group clients.

In [12,33], BA is applied in the second phase of 2PC. But in fact, a single malicious participant can paralyze the system by simply voting "NO", which can be tolerated in our approaches.

If we assume Byzantine faults, the problem of distributed atomic operations is a Weak Byzantine Generals Problem [19], which is usually solved by Byzantine agreement techniques [19,20] theoretically. However, Byzantine agreement algorithms are usually not affordable in distributed environment due to their high communication costs.

6.3. Application level intrusion tolerance

Because our approaches are based on the state machine approach, they cannot handle consistent but malicious requests sent by malicious clients. Transaction level intrusion tolerance and attack recovery techniques have been studied in [5,47,48]. These work implements intrusion tolerance at the transaction level. When intrusions are detected by an intrusion detector, the database system isolates and confines the impaired data. Then, the system can recover from malicious transactions. These work is different from ours in that their system has a "window" from the status of destruction to the status of recovery. And user may obtain wrong data during this window. Hence, their systems cannot totally mask the intrusions. Nevertheless, these techniques and our intrusion masking distributed computing models are complementary to each other, and these techniques can be integrated into our scheme.

6.4. Voting after reliable atomic group communications

Distributed operations implemented by voting after reliable atomic group communications have been proposed in [16,36]. Their work can be modeled by Fig. 16.

Suppose there is a single step operation that a size-m-group $\mathscr{G}$ sends a request to a size-n-group $\mathscr{G}'$. In G-AC configuration (see Fig. 9(b)), there are $mn + f(n)$ messages that need to send, where $f(n)$ is the messaging cost in MTO per consensus. By voting after reliable atomic group communication, there are totally $mg(n)$ messages that need to send, where $g(n)$ is the messaging cost in TO per consensus. Usually

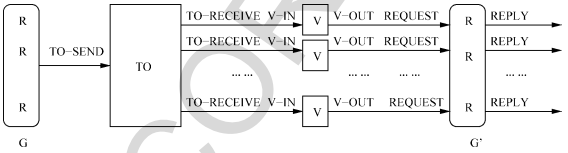


Fig. 16. Voting after reliable group communication.

$f(n) \approx g(n)$ and $g(n) \gg n$ (e.g., in BFT, $g(n) = 2n^2 - n - 1$). So, their approach costs about $(m-1)g(n)$ more messages than our V-MTO-R model without improving survivability.

Finally, this work is different from our previous work [46] as follows. (a) In this paper, we show how a set of well specified I/O automatons can be composed to construct intrusion masking models, while in [46], no automatons are specified and no composite intrusion masking models are proposed. (b) In [46], a specific BFT-S based intrusion masking approach is proposed, but it is not generic. In this paper, we identify the building blocks for intrusion masking distributed computing and these building blocks are simple and powerful enough to compose new models besides the two models we proposed. (c) We have more results of analysis of survivability and efficiency.

7. Conclusions

Today, many critical applications want their services to be executed in an intrusion-masking fashion. Compared with intrusion tolerance techniques, where some integrity or availability degradations are usually caused, intrusion masking techniques use substantial replications to avoid such degradations. Existing intrusion masking techniques can effectively mask intrusions when processing requests from a client using a server replica group, but they are fairly limited in processing distributed operations across multiple server replica groups. As more and more applications need to process distributed operations in an intrusion-masking fashion, it is in urgent need to overcome the limitations of existing intrusion masking techniques.

In this paper, we specify and compose two intrusion-masking models for inter-replica-group distributed computing. Using these two models, a variety of applications can mask intrusions. Our intrusion masking models overcome the limitations of existing intrusion masking techniques. The survivability of our intrusion-masking models is quantitatively analyzed. A simple yet practical implementation method of our intrusion-masking models is proposed and applied to build two intrusion-masking two-phase-commit (2PC) protocols, and the corresponding efficiency is analyzed. The two intrusion-masking 2PC protocols and the analysis results show that the proposed intrusion-masking models have good utility, practicality, and survivability. Finally, the composition methodology developed in this paper can also be used to develop other intrusion-masking distributed computing models.

Acknowledgements

We thank anonymous reviewers for their valuable and insightful comments. This work is supported by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Material Command, USAF, under agreement number F30602-00-2-0575, by DARPA and AFRL, AFMC, USAF, under award number F20602-02-1-0216, by NSF CCR-TC-0233324, and by Department of Energy Early Career PI Award.

References

[1] D.A. Agarwal, L.E. Moser, P.M. Melliar-Smith and R.K. Budhia, The totem multiple-ring ordering and topology maintenance protocol, *ACM Transactions on Computer System* **16**(2) (1998), 93–132.

[2] D. Agrawal and A. EL Abbadi, The generalized tree quorum protocol: An efficient approach for managing replicated data, *ACM Transactions on Database Systems* **17**(4) (1992), 689–717.

[3] L. Alvisi, D. Malkhi, E. Pierce and M.K. Reiter, Fault detection for byzantine quorum systems, *IEEE Transactions on Parallel and Distributed Systems* **12**(9) (2001), 996–1007.

[4] Y. Amir, L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal and P. Ciarfella, The totem single-ring ordering and membership protocol, *ACM Transactions on Computer System* **13**(4) (1995), 311–342.

[5] P. Ammann, S. Jajodia and P. Liu, Recovery from malicious transactions, *IEEE Transaction on Knowledge and Data Engineering* **14**(5) (2002), 1167–1185.

[6] A. Arora and S.S. Kulkarni, Designing masking fault-tolerance via nonmasking fault-tolerance, *IEEE Transactions on Software Engineering* **24**(6) (1998), 435–450.

[7] M. Castro and B. Liskov, Practical byzantine fault tolerance, in: *The Third Symposium on Operating Systems Design and Implementation (OSDI '99)*, New Orleans, USA, 1999, pp. 173–186.

[8] D. Chen, S. Garg and K.S. Trivedi, Network survivability performance evaluation: a quantitative approach with applications in wireless ad-hoc networks, in: *Proceedings of the 5th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, ACM Press, 2002, pp. 61–68.

[9] S.-W. Chen and C. Pu, A structural classification of integrated replica control mechanisms, Technical Report CUCS-006-92, Columbia University, New York, NY, 1992.

[10] M.J. Fischer, N.A. Lynch and M.S. Paterson, Impossibility of distributed consensus with one faulty process, *Journal of the ACM* **32**(2) (1985), 374–382.

[11] G.R. Ganger, P.K. Khosla, M. Bakkaloglu, M.W. Bigrigg, G.R. Goodson, S. Oguz, V. Pandurangan, C.A.N. Soules, J.D. Strunk and J.J. Wylie, Survivable storage systems, in: *DARPA Information Survivability Conference and Exposition*, Vol. 2, Anaheim, CA, IEEE, 2001, pp. 184–195.

[12] H. Garcia-Molina and F. Pittelli, Applications of byzantine agreement in database systems, *ACM Transactions on Database Systems* **11**(1) (1986), 27–47.

[13] F.C. Gärtner, Fundamentals of fault-tolerant distributed computing in asynchronous environments, *ACM Computing Surveys* **31**(1) (1999), 1–26.

[14] S. Jha and J.M. Wing, Survivability analysis of networked systems, in: *Proceedings of the 23rd International Conference on Software Engineering*, IEEE Computer Society, 2001, pp. 307–317.

[15] B. Kemme and G. Alonso, A new approach to developing and implementing eager database replication protocols, *ACM Transactions on Database Systems* **25**(3) (2000), 333–379.

[16] B. Kemme, F. Pedone, G. Alonso and A. Schiper, Processing transactions over optimistic atomic broadcast protocols, in: *Proc. 19th IEEE International Conference on Distributed Computing Systems*, Austin, TX, USA, 1999, pp. 424–431.

[17] K.P. Kihlstrom, L.E. Moser and P.M. Melliar-Smith, The securering group communication system, *ACM Transactions on Information and System Security* **4**(4) (2001), 371–406.

[18] K. Kyandoghere, Survivability performance analysis of rerouting strategies in an atm/vp dcs survivable mesh network, *ACM SIGCOMM Computer Communication Review* **28**(5) (1998), 22–49.

[19] L. Lamport, The weak byzantine generals problem, *Journal of the Association for Computing Machinery* **30**(3) (1983), 668–676.

[20] L. Lamport, R. Shostak and M. Pease, The byzantine general problem, *ACM Transactions on Programming Languages and Systems* **4**(3) (1982), 382–401.

[21] W. Lee and S.J. Stolfo, A framework for constructing features and models for intrusion detection systems, *ACM Transactions on Information and System Security* **3**(4) (2000), 227–261.

[22] H.F. Lipson and D.A. Fisher, Survivabilityąla new technical and business perspective on security, in: *Proceedings of the 1999 Workshop on New Security Paradigms*, ACM Press, 2000, pp. 33–39.

[23] P. Liu and S. Jajodia, Multi-phase damage confinement in database systems for intrusion tolerance, in: *Proc. 14th IEEE Computer Security Foundations Workshop*, Nova Scotia, Canada, 2001, pp. 191–205.

[24] P. Liu, S. Jajodia and C.D. McCollum, Intrusion confinement by isolation in information systems, *Journal of Computer Security* **8**(4) (2000), 243–279.

[25] P. Liu, Architectures for intrusion tolerant database systems, in: *The 18th Annual Computer Security Applications Conference*, 2002, pp. 311–320.

[26] X. (Sean) Liu and W. Du, Multiview access protocols for large-scale replication, *ACM Transactions on Database Systems* **23**(2) (1998), 158–198.

[27] N. Lynch and M.R. Tuttle, An introduction to input/output automata, *CWI Quarterly* **2**(3) (1989), 219–246; also available as MIT Technical Memo MIT/LCS/TM-373.

[28] N. Lynch, *Distributed Algorithm*, Morgan Kaufmann, San Mateo, CA, 1996.

[29] D. Malkhi and M. Reiter, A high-throughput secure reliable multicast protocol, *Journal of Computer Security* **5**(1) (1997), 113–127.

[30] D. Malkhi and M. Reiter, Byzantine quorum system, *Distributed Computing* **11**(4) (1998), 203–213.

[31] D. Malkhi and M.K. Reiter, Secure and scalable replication in phalanx (extended abstract), in: *The Seventeenth IEEE Symposium on Reliable Distributed Systems*, 1998, pp. 51–58.

[32] D. Malkhi and M.K. Reiter, An architecture for survivable coordination in large distrusted systems, *IEEE Transactions on Knowledge and Data Engineering* **12**(2) (2000), 187–202.

[33] C. Mohan, R. Strong and S. Finkelstein, Method for distributed transaction commit and recovery using byzantine agreement within clusters of processors, in: *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, 1983, pp. 89–103.

[34] G. Morgan and P.D. Ezhilchelvan, Policies for using replica groups and their effectiveness over the Internet, in: *The Second International COST264 Workshop on Networked Group Communication (NGC 2000)*, Palo Alto, California, 2000, pp. 119–129.

[35] L.E. Moser and P.M. Melliar-Smith, Byzantine-resistant total ordering algorithms, *Journal of Information and Computation* **150**(1) (1999), 75–111.

[36] P. Narasimhan, L.E. Moser and P.M. Melliar-Smith, Strong replica consistency for fault-tolerant corba applications, in: *Sixth International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '01)*, Rome, Italy, 2001, pp. 10–17.

[37] M. Tamer Özsu and P. Valduriez, *Principles of Distributed Database Systems*, Prentice Hall, 1999.

[38] M.K. Reiter, Secure agreement protocols: Reliable and atomic group multicast in rampart, in: *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, Fairfax, VA, 1994, pp. 60–80.

[39] F.B. Schneider, Implementing fault-tolerant services using the state machine approach: a tutorial, *ACM Computing Surveys* **22**(4) (1990), 299–319.

[40] V. Stavridou, B. Dutertre, R.A. Riemenschneider and H. Saidi, Intrusion tolerant software architectures, in: *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01, Proceedings*, Vol. 2, 2001, pp. 230–241.

[41] J.D. Strunk, G.R. Goodson, M.L. Scheinholtz, C.A.N. Soules and G.R. Ganger, Self-securing storage: Protecting data in compromised systems, in: *Operating Systems Design and Implementation*, San Diego, CA, USENIX Association, 2000, pp. 165–180.

[42] O. Wolfson, S. Jajodia and Y. Huang, An adaptive data replication algorithm, *ACM Transactions on Database Systems* **22**(2) (1997), 255–314.

[43] J.J. Wylie, M. Bakkaloglu, V. Pandurangan, M.W. Bigrigg, S. Oguz, K. Tew, C. Williams, G.R. Ganger and P.K. Khosla, Selecting the right data distribution scheme for a survivable storage system, Technical Report CMU-CS-01-120, Carnegie Mellon University, 2001.

[44] J.J. Wylie, M.W. Bigrigg, J.D. Strunk, G.R. Ganger, H. Kiliccote and P.K. Khosla, Survivable information storage systems, *IEEE Computer* **33**(8) (2000), 61–68.

[45] C.T.Y. Amir, From total order to database replication, in: *22nd International Conference on Distributed Computing Systems (ICDCS '02)*, Vienna, Austria, 1992, pp. 494–503.

[46] M. Yu, P. Liu and W. Zang, Intrusion masking for distributed atomic operations, in: *The 18th IFIP International Information Security Conference*, Athens Chamber of Commerce and Industry, Greece, IFIP Technical Committee 11, Kluwer-Academic, 2003, pp. 229–240.

[47] M. Yu, P. Liu and W. Zang, Self-healing workflow systems under attacks, in: *The 24th International Conference on Distributed Computing Systems (ICDCS '04)*, 2004, pp. 418–425.

[48] M. Yu, P. Liu and W. Zang, Multi-version based attack recovery of workflow, in: *The 19th Annual Computer Security Applications Conference*, 2003, pp. 142–151.