Various Interprocess Communication Checkpointing Algorithms and Security Aspects In Distributed Systems

Prateek Sharma*
Indraprastha Institute of Information Technology-Delhi, New Delhi, India
* E-mail: prateek1215@iiitd.ac.in

Abstract

This paper focuses on various interprocess communication checkpointing algorithms in distributed systems and also presents a comparative study of security issues associated with distributed systems. First, we study each algorithm in detail one by one and see how these algorithms help in determining global states of a distributed system and achieving mutual exclusion and how they are different from each other. We evaluate the performance of each algorithm on some basic parameters like complexity, features, types of channel used, etc.

In the later half of the paper, we present a comparative study of security issues associated with distributed systems. We give a brief idea of various security threats like eavesdropping, Denial of Service (DOS) attack, replaying, message tampering, etc. and discuss authentication in detail.

Categories and Subject Descriptors: C.2.4 [Computer-Communication Networks]: Distributed Systems-distributed applications; distributed databases; network operating systems; D.4.1 [Operating Systems]: Process Management-concurrency; deadlocks, multiprocessing/multiprogramming; mutual exclusion; scheduling; synchronization; D.4.5 [Operating Systems]: Reliability-backup procedures; checkpoint/restart; fault-tolerance; verification

General Terms: Algorithms
Additional Key Words and Phrases: Global States, Distributed deadlock detection, distributed systems, message communication systems

1 Introduction

This paper presents various checkpointing algorithms used in a distributed system to determine global states in distributed system and to achieve mutual exclusion [9, 16–18]. In a distributed system, process communicates in two ways- by sending and receiving messages and by shared memory. In this paper, we only focus on communication using message passing. A process can record its own state and states of it’s corresponding channels. To determine global state of a distributed system, a process p needs the cooperation of other processes that records their global states and send the recorded states to p. A global state of a distributed system is consistent if no transactions are in progress. All processes can record their local states at the same instant of time unless there is a notion of common clock. For consistent global states, we need a consistent global checkpoint. A global checkpoint is a transaction which must view a globally consistent system state for correct operation.

Later, we present security issues related to distributed systems. Recent work examining distributed system security is evaluated in a detailed and analytical way. An analysis of distributed system security requires identification of those factors and issues that affect the trustworthiness of distributed systems.
We studied briefly main attacks on distributed systems such as eavesdropping, masquerading, Denial of Service attacks and explained the authentication attack in detail. Authentication attacks are the most frequent attacks on distributed system security as compare to other attacks. Getting access to unauthorized material without permission can cause a lot of potential loss of resources.

2 Interprocess Communication Checkpointing Algorithms And Distributed System

2.1 Architecture of Distributed System

Distributed system consists of processes and channels. A process is defined by a set of states and a set of events. An event e in a process p is an action that can change the state of the process and the state of one channel incident on it. A global state of a distributed system is a set of component processes and states of channel (Figure 1). The occurrence of event may change the global state of the system. We define an event e by the 5-tuple \(< p, s, s', M, C >\) where p is a process, s is the state of p before the event and s’ is the state of p after the event. M is a message sent or received along channel c. An event e can occur in a global state S if the state of the process p in global state is s. We define a function called next where \(next < S, e >\) is the global state immediately after the occurrence of event e in global state S. The value of \(next < S, e >\) is defined only if event can occur in global state S, in which case \(next < S, e >\) is the global state identical to S [8].

2.2 Consistent Global State

To understand consistency in global states, first we have to understand the term cut. A cut in a space-time diagram is a line joining an arbitrary point at each process that slices the space-time diagram into two parts, past and future. A consistent global state corresponds to a cut in which every message received in the past of the cut was sent in the past of that cut. Such a cut is known as consistent cut. For example in Figure 2, cut C1 is inconsistent because message m1 is flowing from the future to the past. On the other hand, cut C2 is consistent and message m2 must be captured in the state of the channel \(m_{21}\).
2.3 Overheads of Checkpointing Algorithms

**Coordination Overhead:**

In distributed systems, coordination among all the processes is necessary in order to obtain a consistent global state. Special messages and piggybacked information with regular messages are used in obtaining coordination among processes. Special messages and piggybacked information causes coordination overhead. The bookkeeping operations that are necessary to maintain the coordination also causes coordination overhead.

**Context-Saving Overhead:**

The time taken to save the global context of a computation is called context saving overhead. This overhead is directly proportional to the size of context, that is, larger the size of context larger is the context saving overhead and vice versa. The context overhead is transferred to the network if storage is not available with every node in a multiprocessor system. Network transmission delay also contributes to the overhead.

2.4 Checkpointing Algorithms For Communication

2.4.1 Chandy and Lamport Algorithm

The Chandy-Lamport algorithm uses a control message called a *marker* whose role in a distributed system is to separate messages in the channel. After a site has recorded its snapshot, it sends a *marker* along all the outgoing channels before sending out any more messages. A *marker* separates the messages in the channel into those to be included in the snapshot from those not to be recorded in the snapshot. A process must record its snapshot no later than what it receives a *marker* on any of its incoming channels [11].

The algorithm can be initiated by any process by executing the "Marker Sending Rule" in which a state recorded its local state and sends a *marker* on each outgoing channel. A process executes the "Marker Receiving Rule" on receiving a *marker*. If the process has not yet recorded its local state, it records the state of the channel on which the *marker* is received as empty and executes the "Marker Sending Rule" to record its local state. The algorithm terminates after each process has received a *marker* on all of its incoming channels. All the local snapshots get spread widely to all other process and all the processes can determine the global state.

**Algorithm:**

![Space-Time Diagram of Consistent Global State](image)
**Marker Sending Rule For Process p:**

Process p record its state. For each outgoing channel c on which marker has not been sent, p sends a marker along c before p sends further messages along c.

**Marker Receiving Rule For Process p':**

On receiving a marker along channel c-
- if process p’ does not recorded its state then
  - begin Record the state of c as empty set
  - Follow the ”Marker Sending Rule”
  - end
- else Record the state of c as the set of messages received along c after p’ state was recorded and before p’ received the marker along c.

**Modifications in the features of the Chandy-Lamport Algorithm**

There are certain aspects of Chandy-Lamport Algorithm that can be improved:-

**FIFO vs. non-FIFO Channels**

Chandy-Lamport algorithm works on FIFO channels. In other words, if a message m1 is sent by the process pi before it sends out another message m2 to pj, then, m1 must reach before m2 reaches pj. Advantages of FIFO channel is the sequencing of messages without using any explicit sequence number. Non-FIFO channels needs headers to correctly sequencing the messages. The possibility of non-FIFO channels is justified in the case of distributed systems as messages routed through different routes before reaching destination and may be out of order after reaching the destination.

**Centralized vs. Distributed Checkpoint Initiation**

In a centralized algorithm like Chandy-Lamport, there is always a master node that initiates the checkpoints and coordinates the participating nodes. Disadvantage of this technique is that every node has to initiate the checkpoints when master node decides to checkpoint. Nodes can be given the freedom in initiating checkpoints by allowing any node in the system to initiate checkpoints. [4]

**Static vs. Dynamic Checkpointing**

Chandy and Lamport algorithm does not assume any specific knowledge about the programs being executed. At runtime, the checkpointing algorithm is initiated and the checkpoints are taken. It is known as dynamic checkpointing algorithm. An alternative to this approach is to identify the checkpointing locations statically before executing the programs. Static approach is widely used in uniprocessor checkpointing.

**Periodic vs. non-Periodic Checkpointing**

Periodic checkpointing algorithm ensures that the maximum information loss cannot exceed the period between consecutive global checkpoints. Aperiodic algorithms do not force the nodes to
initiate checkpoints at predetermined times. Aperiodic algorithms are helpful in situations where nodes should not be interrupted for checkpointing at certain time instances. Aperiodic algorithms are also helpful if advancing or delaying the checkpointing process minimises the context-saving overhead [5]. The total cost incurred in the aperiodic algorithms is once again in the terms of constructing the global consistent state. Non-periodic algorithms will require nodes to initiate the checkpoint independently and so all the problems of independent checkpointing algorithm will be part of this option.

2.4.2 Mattern’s Algorithm

Mattern’s algorithm is based on vector clocks and assumes a single initiator process. In this algorithm, the initiator ticks its local state and select a future vector time $s$ at which it would like a global snapshot to be recorded. It then broadcast this time $s$ and freezes all activity until it receives all acknowledgements of the receipt of this broadcast. When a process receives the broadcast, it remembers the value $s$ and returns an acknowledgement to the initiator. After have received an acknowledgement from every process, the initiator increases its vector clock to $s$ and broadcasts a dummy message to all processes. The value of this dummy message forces each recipient to increases its clock to a value $\geq s$ if not already $\geq s$. Each process takes a local snapshot and sends it to the initiator when (just before) its clock increases from a value less than $s$ to a value $\geq s$. The state of $C_{ij}$ (where C is a channel from process i to process j) is all messages sent along $C_{ij}$, whose timestamp is smaller than $s$ and which are received by pj after recording $LS_j$ [10].

A termination detection scheme for non-FIFO channels is required to detect that no white messages are in transit. One of the following two schemes can be used for termination detection:

**First Scheme:**

Each process i keeps a counter $ctr_i$ that indicates the difference between the number of white messages it has sent and received before recording its snapshot. It reports this value to the initiator process along with its snapshot and forwards all white messages, it receives henceforth, to the initiator. The snapshot collection terminates when the initiator has received $\sum_i ctr_i$ number of forwarded white messages.

**Second Scheme:**

Each red message sent by a process carries a piggybacked value of the number of white messages sent on that channel before the local state recording. Each process keeps a counter for the number of white messages received on each channel. A process can detect termination of recording the states of incoming channels when it receives as many white messages on each channel as the value piggybacked on red messages received on that channel.

2.4.3 Ricart-Agrawala Algorithm

The Ricart-Agrawala algorithm assumes the communication channels are FIFO. The algorithm uses two types of messages: REQUEST and REPLY.

**Algorithm Outline:**
A process sends a REQUEST message to all other processes to request their permission to enter the critical section. A process sends a REPLY message to a process to give its permission to that process. Processes use Lamport-style logical clocks to assign a timestamp to critical section requests and timestamps are used to decide the priority of requests. Each process $p_i$ maintains the request-Deferred array $RD_i$, the size of which is the same as the number of processes in the system.

Initially, $\forall i \forall j: RD_i[j] = 0$. Whenever $p_i$ defer the request sent by $p_j$, it sets $RD_i[j] = 1$ and after it has sent a REPLY message to $p_j$, it sets $RD_i[j] = 0$.

Algorithm -

**Requesting The Critical Section:**

When a site $S_i$ wants to enter CS, it broadcasts a timestamped Request message to all other sites. When site $S_j$ receives a Request message from site $S_i$, it sends a reply message to site $S_i$ if site $S_j$ is neither requesting nor executing the CS, or if the site $S_j$ is requesting and $S_i$’s request’s timestamp is smaller than site $S_j$’s own request’s timestamp. Otherwise, the reply is deferred and $S_j$ sets $RD_j[i] = 1$.

**Executing The Critical Section:**

Site $S_i$ enters the CS after it has received a REPLY message from every other site it sent an REQUEST message to.

**Releasing The Critical Section:**

When site $S_i$ exits the CS, it sends all the deferred REPLY messages: $\forall j$ if $RD_i[j] = 1$, then sends a REPLY message to $S_j$ and set $RD_i[j] = 0$.

### 2.4.4 Maekawa’s Algorithm

Maekawa’s algorithm was the first quorum-based mutual exclusion algorithm. The request set for sites (i.e. quorums) in Maekawa’s algorithm are constructed to satisfy the following conditions-

- **M1:** $(\forall i \forall j: i \neq j, 1 \leq i, j \leq N:: R_i \cap R_j \neq \phi)$
- **M2:** $(\forall i: 1 \leq i \leq N:: S_i \in r_i)$
- **M3:** $(\forall i: 1 \leq i \leq N:: \mod R_i = k)$
- **M4:** Any site $S_j$ is contained in $k$ number of $R_i, 1 \leq i, j \leq N$.

Maekawa used the theory of projective planes and showed that $N = k(k - 1) + 1$. This relation gives $\mod r_i = \sqrt{N}$. Conditions M1 and M2 are necessary for correctness; whereas conditions M3 and M4 provide other desirable features to the algorithm. Condition M3 states that the size of the requests sets of all sites must be equal implying that all sites should have to do equal amount of work to invoke mutual exclusion. Condition M4 enforces that exactly the same number of sites should request permission from any site implying that all sites have ”equal responsibility” in granting permission to other sites.

Algorithm:

A site $S_i$ executes the following steps to execute the CS.
**Requesting The Critical Section:**

When a site $S_i$ requests access to the CS by sending REQUEST(i) messages to all sites in its request set $R_i$. When site $S_j$ receives a REQUEST(i) message from site $S_i$, it sends a REPLY(j) message to site $S_i$ provided it hasn’t sent a REPLY message to a site since its receipt of the last RELEASE message. Otherwise, it queues up the REQUEST(i) for later consideration [20].

**Executing The Critical Section:**

Site $S_i$ executes the CS after it has received a REPLY message from every site in $R_i$.

**Releasing The Critical Section:**

After the execution of the CS is over, site $S_i$ sends a RELEASE(i) message to every site in $R_i$. When a site $S_j$ receives a RELEASE(i) message from site $S_i$, it sends a REPLY message to the next site waiting in the queue and deletes that entry from the queue. If the queue is empty, then the site updates its state to reflect that it has not sent out any REPLY message since the receipt of the last RELEASE message.

### 2.4.5 Suzuki-Kasami Broadcast Algorithm

If a site wants to enter the CS and it does not have the token, it broadcasts a REQUEST message for the token to all the other sites. A site which possesses the token sends it to the requesting site upon the receipt of its REQUEST message. If a site receives a REQUEST message when it is executing the CS, it sends the token only after it has completed the execution of the CS.

This algorithm addresses the following two design issues:

1. **How to distinguish an outdated REQUEST message from a current REQUEST message:**
   Due to variable message delays, a site may receive a token request message after the corresponding request has been satisfied. If a site cannot be determined if the request corresponding to a token request has been satisfied, it may dispatch the token to a site that does not need it. This will not violate the correctness, however, this may seriously degrade the performance.

2. **How to determine which site has an outstanding request for the CS:**
   After a site has finished the execution of the CS, it must determine what sites have an outstanding request for the CS so that the token can be dispatched to one of them.

The first issue is addressed in the following manner:

A REQUEST message of site $S_j$ has the form REQUEST(j,n) where $n(n = 1, 2, 3, \ldots)$ is a sequence number which indicates that site $S_j$ is requesting its $n^{th}$ CS execution. A site $S_i$ always keeps an array of integers $RN_i[1..N]$ where $RN_i[j]$ denotes the largest sequence number received in a REQUEST message so far from site $S_j$. When site $S_i$ receives a REQUEST(j,n) message, it sets $RN_i[j] := max(RN_i[j], n)$. When a site $S_i$ receives a REQUEST(j,n) message, the request is going to be outdated if $RN_i[j] > n$. 
The second issue is addressed in the following manner-

The token consists of a queue of requesting sites, $Q$, and an array of integers $LN[1..N]$, where $LN[j]$ is the sequence number assigned to the request which has been executed most recently by site $S_j$. After the execution of CS by a site $S_i$, it updates $LN[i] = RN_i[i]$ to indicate the site that its request has been executed corresponding to the sequence number $RN_i[i]$. If $RN_i[j] = LN[j] + 1$ at site $S_i$, then a token is currently being requested by site $S_j$.

Algorithm:

Requesting the critical section-

If requesting site $S_i$ does not have the token, then it can increment its sequence number, $RN_i[i]$ and sends a REQUEST message to all the other sites in the system. Here 'sn' is the updated value of $RN_i[i]$. When a site $S_j$ receives this message, it sets

$$RN_j[i] = \max(RN_j[i], sn)$$

If site $S_j$ has an idle token, then it sends the token to $S_i$ if

$$RN_j[i] = LN[i] + 1$$

Executing the critical section-

Site $S_i$ executes the CS after it has received the token.

Releasing the critical section-

After finishing the execution of CS, following are the actions taken by site $S_i$:

It sets $LN[i]$ element of the token array equal to $RN_i[i]$. For every site $S_j$ whose id is not in the token queue, it appends its id to the token queue if $RN_i[j] = LN[j] + 1$. If after the above update token queue is empty, site $S_i$ deletes the top site id from the token queue and sends the token to the site indicated by id.

2.4.6 Lai-Yang Algorithm

The Lai-Yang algorithm fulfills the role of a marker in a non-FIFO system using a coloring on computation messages. In this algorithm, every process is white in initial state and it turns red while taking a snapshot. The equivalent of a "Marker Sending Rule" is executed when the process turns red. Every message sent by a white process is colored red and similarly every message sent by a red process is colored white. A white(red) message is a message that was sent before(after) the sender of that message recorded its local snapshot. Every white process takes its snapshot at its convenience, but not later than the instant it receives a red message [19]. Every white process keeps records of the white messages sent or received along each channel. When a process turns red, it sends its records along with its snapshot to the initiator.
process that collects the global snapshot. The initiator process evaluates $\text{transit}(LS_i, LS_j)$ to compute the state of a channel $C_{ij}$ as given below:

$$SC_{ij} = \text{white messages sent by } p_i \text{ on } C_{ij} - \text{white messages received by } p_j \text{ on } C_{ij}$$

$$= \text{send}(m_{ij}) \cup \text{send}(m_{ij} \epsilon LS_i) - \text{rec}(m_{ij}) \cup \text{rec}(m_{ij} \epsilon LS_j)$$

### 2.4.7 Spezialetti-Kearns Algorithm

There are two phases in obtaining a global snapshot. First, locally recording the snapshot at every process and second is distributing the resultant global snapshot to all the initiators.

**Efficient snapshot recording:**

In the Spezialetti-Kears algorithm, a *marker* carries the identifier of the algorithm’s initiator. Each process has a variable *master* to keep track of the initiator of the algorithm. A *region* in the system encompasses all of the processes whose *master* field contains the identifier of the same initiator. When the initiator’s identifier in a *marker* received along a channel is different from the value in the *master* variable, the sender of the *marker* lies in a different *region*. The identifier of the concurrent initiator is recorded in a local variable *idSet*. The state of the channel is recorded in the similar fashion as it is recorded in the Chandy-Lamport algorithm (including those that cross a border between *regions*). Snapshot recording at a process is complete after it has received a *marker* along all its channels. After every process has recorded its snapshot, the system is partitioned into as many *regions* as the number of concurrent initiations of the algorithm. Variable *idSet* at a process contains the identifiers of the neighbouring *regions* [15].

**Efficient distribution of the recorded snapshot:**

In the snapshot recording phase, a forest like structure of spanning trees is created in the system. The initiator of the algorithm is the root of the spanning tree and all the processes in its *region* belongs to its spanning tree. If a process $p_i$ receives its first *marker* from process $p_j$, then process $p_j$ is the parent of the process $p_i$ in the spanning tree. When an intermediate process in a spanning tree has received the recorded states from all of its child processes and has recorded the states of all the incoming channels, it forwards its locally recorded state and the locally recorded states of all its descendant processes to its parent. The initiator exchanges the snapshot of its *region* with the initiators in adjacent *regions* in rounds. The message complexity of snapshot recording algorithm is $O(e)$ irrespective of the number of concurrent initiations of the algorithm.

### 2.5 Performance Evaluation

In this subsection, we evaluate the performances of various algorithms that we have discussed so far in this paper on some basic parameters like complexity, features, type of channel used and type of approach used in the algorithm. We build a comparison table (i.e., Table 1) showing comparison of different snapshots algorithms in distributed systems. As we can conclude from the table that some algorithms uses FIFO channels whereas other algorithms uses Non-FIFO and bidirectional channels. We can also conclude that message complexity of Chandy-Lamport algorithm is more than Spezialetti-Kears Algorithm. Similarly, we can conclude other things from the table and compare their performances. The algorithms are simple and easy to implement and requires channels to be in FIFO, non-FIFO and bidirectional modes to be
<table>
<thead>
<tr>
<th>S.no.</th>
<th>Algorithm</th>
<th>Complexity</th>
<th>Features</th>
<th>Channel</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Chandy and Lamport Algorithm</td>
<td>Message complexity is $O(e)$ and $O(d)$ is time where $d$ is diameter of graph</td>
<td>Basic algorithm for global snapshot algorithm</td>
<td>FIFO</td>
<td>Centralized</td>
</tr>
<tr>
<td>2.</td>
<td>Mattern Algorithm</td>
<td>Total no of message is $O(</td>
<td>c</td>
<td>)$, response time is $2n$, total message space is $O(n^2)$</td>
<td>No message history required</td>
</tr>
<tr>
<td>3.</td>
<td>Ricart-Agrawala Algorithm</td>
<td>The no of messages required per CS is $2*(n-1)$</td>
<td>Reduce the number of messages to REQUEST and REPLY w.r.t Chandy-Lamport</td>
<td>Non-FIFO</td>
<td>Centralized</td>
</tr>
<tr>
<td>4.</td>
<td>Maekawa Algorithm</td>
<td>The no of messages required per CS is $k*\sqrt{n}$ where $3 \leq k \leq 6$</td>
<td>Process enters critical section if it succeeds in acquiring locks from its entire quorum</td>
<td>FIFO</td>
<td>Centralized</td>
</tr>
<tr>
<td>5.</td>
<td>Suzuki-kasami Broadcast Algorithm</td>
<td>$0$ to $n$ messages for CS invocation</td>
<td>Not based on Lamport logical clock, uses sequence numbers instead</td>
<td>FIFO</td>
<td>Centralized</td>
</tr>
<tr>
<td>6.</td>
<td>Lai-Yang Algorithm</td>
<td>$O(</td>
<td>c</td>
<td>), here c is the set of channels in the system</td>
<td>Marker systems, use Markers piggy-backed on messages</td>
</tr>
<tr>
<td>7.</td>
<td>Spezialetti-Kearns Algorithm</td>
<td>Message complexity of recording is $O(e)$ and assembling and distributing the snapshot is $O(rn^2)$</td>
<td>Two Phase Algorithm, support concurrent initiators, advancement of Lamport-Chandy Algorithm</td>
<td>Bidirectional</td>
<td>Centralized</td>
</tr>
</tbody>
</table>

- $e$: the number of edges in the graph
- $n$: the number of processes
- $r$: the number of concurrent initiations

**Table 1.** Comparison of Different Snapshots Algorithms in Distributed Systems
acknowledged. Each algorithm has distinct and unique features when compared to other algorithms and some algorithms have specific requirements to get implemented. The algorithms also require some amount of storage space to store messages that they sent and received from other processes [14].

3 Security Aspects in Distributed System

An important aspect of distributed systems is security. Secure communication between processes in distributed system is the main concern. A secure channel provides confidentiality in the data sent over that channel and assures that information will not be revealed to third-party users without the permission of the information’s owner. This requires determining the identities of the entities involved in the system and by keeping track of which resources they are allowed to access. An insecure distributed system is always vulnerable to the attacks and can present confidential information of the users to the attackers when attacked. Securing the distributed system should be the top priority because in a distributed system, if one component is compromised it may affect the security of all other components of the distributed system. [2]

3.1 Threats

Confidentiality and integrity are not difficult to achieve in a system when no one attempts to perform unauthorised access. The main concern for security arises when someone tries to gain access to a resource not allotted to it in an inappropriate manner. The goal of security in distributed system is to protect the services and data against security threats.

We basically distinguish security threats in four categories:

**Interception:** An unauthorised identity is trying to gain access to a resource not allotted to it. Examples are eavesdropping, copying data, breaking into the system, etc.

**Modification:** Tempering or modification of the contents of data without the permission or knowledge of the owner of data comes under this category. Examples are intercepting and changing transmitted data.

**Interruption:** Making services unavailable and unusable for the user at real-time comes under this category, etc. Examples are denial of service attacks, deletion and corruption of data.

**Fabrication:** Generation of additional data or activity that actually does not exist comes under this category. Examples are adding entry to a password database, replaying previous sent messages.

The last three threats include some form of providing false data. They provide false information to the system user.

3.2 Attacks

There are basically two forms of attacks in distributed system: *active* attacks and *passive* attacks. Active attacks include altering/modification or deletion of data and causing denial of service to authorised users. They represent the modification and fabrication forms of security threats. Active attacks mainly attempt to modify or destroy the files. Active attacks over communication channels attempts to modify the data
sent over that channel.
Passive attacks are mainly based on observation. They do not modify data or temper with services, they present the interception and interruption forms of security threats. The simplest forms of attacks includes *browsing*, which includes examination of all accessible data in non-destructive manner. This leads to the need of confidentiality in the communication protocols. Other indirect attempts are inferring information from traffic analysis, code breaking and so on.

### 3.2.1 Attacking the Communication Channel

Communication channels are the most vulnerable spot in distributed systems for attacks. Many of the attacks faced by the distributed systems come in the form of attacks on the communication channels.

We basically categorised attacks on communication channels in five categories-

**Eavesdropping** attacks involves getting access to data without permission. For example, involves sniffing password sent over the channel [1].

**Message Tampering** attacks involves the modification of messages without the knowledge of owner of data so that they do not do the specific task for which they are originally intended. *Man-in-the-middle* attack is one form of message tampering. In this attack, the attacker intercepts the first message in an exchange of keys and is able to established a secure channel between sender and receiver. Attacker can now view and modify the communication between sender and receiver as he/she placed himself in the middle.

**Masquerading** attacks involves using the identity of other person for sending and receiving messages without their authority. In this attack, attacker sends the messages to the victim with the headers modified so that it looks like that the messages are being sent by a trusted third party.

**Replay** attacks involve resending intercepted messages at a later time in order to repeat the action. This kind of attack can be effective even if communication is authenticated and encrypted.

**Denial of Service** attacks involves the flooding of channel with messages so that access is denied to others.

### 3.2.2 Mechanisms

In order to deal with security threats, distributed system must used some appropriate mechanisms. The four main mechanisms used to protect against security threats are-

**Encryption:** It involves transforming data into some form that an attacker can’t understand. Encryption prevents entities from being able to read or write data that they are not authorised to read or write. Attacker can still be able to access the data but can’t able to do anything with it. Encryption schemes also help to determine if data has been modified.

**Authentication:** It verifies the claimed identity of an entity(a user, client, etc). In order to determine if an entity is authorised to access the particular data or services it is necessary to know who the identity is. Examples of authentication mechanisms include passwords, chipcards and biometric identification.
Authorisation: It determines what actions an authenticated identity is authorised to perform. Authorisation mechanisms are used for tracking who is authorised to access what, and preventing unauthorised access to data and services.

Auditing: It traces which entities access what. Auditing does not directly protect against security threats by preventing attacks, it is necessary in order to determine what went wrong when an attack could not be prevented.

Two lesser mechanisms can also help to improve security if they are used in combination with above mechanisms, however, on their own they provide little benefit.

Obscurity: Obscuring the system implementation details makes finding weakness and attacking the system harder. It is important to note that the underlying mechanisms must be secure for obscurity to work. Expecting a system to secure because its inner workings are obscured provides very weak protection. As soon as details about the system become known, the security gained by obscurity becomes worthless.

Intimidation: This approach often project itself as tough laws against cybercrime and the threat of real world punishments. It is always a good idea to have a good security system in place even if there is a high level of intimidation.

3.3 Authentication

We have discussed briefly four main mechanisms which must be implemented by the distributed system to protect against the security threats. In this paper, we mainly focus on authentication mechanism as identity threats are the frequent and serious attacks on distributed system. Authentication involves verifying the claimed identity of an entity. Depending on the system requirements different levels of strengths of authentication required. For example, in some cases it is enough to provide a user id, while in some other cases a certificate duly signed by the trusted authority may be required to prove identity. In simple terms, authentication is verification plus identification. Identification is the procedure where an entity claims a certain identity while verification is the procedure where that claimed made by entity is checked. Thus the correctness of an authentication relies heavily upon the verification procedure used.

There are mainly three types of authentication of interest in a distributed system-

Message Content Authentication: It verifies whether the content of the message received is same as when it was sent.

Message Origin Authentication: It verifies whether the sender of the received message is the same one who is recorded in the sender field of the message.

General Identity Authentication: It verifies that a identifier’s identity is as claimed.

Message content authentication is commonly handled by tagging a message authentication code (MAC) onto a message before it is sent. Message integrity can be confirmed by recomputing the MAC and comparing it with the one attached to the message on reception of message. Message origin authentication is a subcase of general identity authentication. A successful general identity authentication results in a
belief in verifier that the identifier possesses the claimed identity. General identity authentication is neede
for both authorization and accounting functions.

In a distributed system, authentication is carried out using a protocol that involves message sharing. We refer to these protocols as authentication protocols [13]. Most existing systems used only primitive

![Figure 3. Principles in a Distributed System](image.png)

authentication measures. For example:

Most widespread login procedure requires users to enter their passwords in response to system prompt. Users are one-way authenticated by verifying password against an internally stored table. However, no mechanism let users authenticate the system. Such a design is acceptable only when system is trustworthy.

In a typical client-server interaction, the server on accepting a client request has to trust on the host of the client that it has correctly authenticated the client and the identity provided in the request actually belongs to the client. Such trust is valid only if system’s host are trustworthy and communication channels are secure.

In a distributed system, the entities that require authentication are hosts, users and processes. They constitute the principals of authentication (Figure 3).

**Hosts:** Hosts are the entities which are addressed at the network level. A host is distinguished from the underlying supporting hardware. For example, a host H working on a workstation W can be moved to work on any other workstation W’ by performing the bootstrap sequence of H on W’. Hosts are usually identified by its network address or home whereas sequence number is used to identify a particular host hardware.

**Users:** Users are the entities that are responsible for all system activities. In other words, users are accountable for all system activities. Most access control and accounting functions are based upon users. Typical users include humans, as well as accounts maintained in the user database.

**Processes:** The system creates processes within the system boundary to represent users. A process requests the resources required for fulfilling the task assigned to that process. Process basically falls into two categories-client and server. Client processes are consumers who obtain services from server processes, who are service providers. A process can act both as a client as well as a server. For example, print servers are usually created by the user root and acts as servers for printing requests by other processes. However, they act as clients while requesting files from file servers.
3.3.1 Authentication Exchanges

Following are the major types of authentication exchanges in a distributed system.

**Host-host:** Host-level activities often require cooperation between hosts. For example, individual hosts exchange link information among them for updating their internal topology maps. In remote bootstrapping, a host must identify a trustworthy bootstrap server to supply the information required for correct initialization.

**User-host:** By logging in a host, a user can gain access in a distributed system. In a system where hosts are scattered over unrestricted areas, a host can be arbitrary compromised, necessitating mutual exclusion between user and the host.

**Process-process:** Two main subclasses exist in this type of authentication exchange—

- **peer-peer communication:** Peer processes must be satisfied with each other’s identity before private communication can begin.

- **client-server communication:** A client request is made only when a client’s identity is affirmed. A client is ready to deliver valuable information to server only after it has verified the server’s identity.

3.3.2 An Authentication Framework

We have so far presented various basic concepts of authentication. In this part, we put these concepts into authentication framework that can be used to design secure distributed system. We described below five aspects of secure distributed system design and the associated authentication needs.

**Host initializations:** Executions of all the processes are taken place inside host. Some hosts (like workstations) allow user logins and thus, behaves like system entry points. The overall security of the distributed system is highly dependent on the security of each host. However in an open network environment, not all hosts can be physically protected. This suggests the importance of host software integrity. One way to compromised a public host is to reboot the host with incorrect initialization information. Authentication can be used to implement secure bootstrapping.

**User logins:** User identity is established at login. All access control decisions and accounting functions are based on this identity. Correct user identification is thus crucial to the functioning of a secure system. Any host in open environment is susceptible to compromise. Therefore a user should not engaged in any activity with the host without first checking the host’s identity.

**Peer communications:** Distributed systems can distribute a task over multiple hosts to achieve a larger throughput. Correctness of such a distributed task depends upon whether process participating in the task can correctly identify the identity of each other. Authentication can be used to distinguish between friend and foe.

**Client-server interactions:** The client-server model provides an attractive model for constructing distributed systems. Servers are willing to provide service only to authorized clients while clients are interested in dealing only with legitimate servers. Authentication can be used to establish a verified consumer-supplier relationship.
Inter-domain communications: Most distributed systems are not centrally owned. For example, a campus-wide distributed system often interconnects individually administered departmental subsystems. Identifying principals across subsystems requires additional authentication mechanisms across domains.

In authentication framework, an authority granting service like the Authentication service need to validate the user login details against the data stored in tabular form in database. A set of service provider interfaces should be define to allow different providers to write their own authentication module by extending this interface. All the clients or application participating in Single Sign-On(SSO) require a handle just after the authentication process is completed and before the system exit to invoke the application specific logic. SSO can be defined as the ability of a user to authenticate once and gain access to variety of web application resources that otherwise would have required individual authentication with each authentication requires different set of credentials.

The Authentication Service should provide the web-based user interface for all authentication modules plugged in the SSO environment. This interface should provide a dynamic means for gathering authentication credentials by presenting the web-based login requirement page to a user requesting access. The Authentication Service should be easy to implement and understand but at the same time not vulnerable to any kind of attacks. The user interface should be highly customizable as per different characteristics of web clients. The Authentication Service must provide the service provider interface(SPI)for the pre and post authentication hooks for customized application. The diagrammatic representation of architecture is shown in Figure 4.

3.3.3 Approaches to Authentication

All authentication procedures include checking the known information about a claimed identity against information acquired from the claimant during the verification procedure. Such checking can be based upon following three approaches-
### Authentication Module Summary

<table>
<thead>
<tr>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize and retrieve the module configuration</td>
</tr>
<tr>
<td>Retrieve the module's credential requirements in the form of callbacks</td>
</tr>
<tr>
<td>Process module specific callbacks for user interaction and submission of required user credentials</td>
</tr>
<tr>
<td>Validate the user credentials</td>
</tr>
<tr>
<td>Get the valid user identifier to be added in the user assertion object</td>
</tr>
<tr>
<td>Get control after successful authentication</td>
</tr>
<tr>
<td>Get control after failure authentication</td>
</tr>
<tr>
<td>Get control on logout</td>
</tr>
</tbody>
</table>

**Proof by Knowledge:** The claimant knows information about the claimed identity that can only be known by a principal with that identity. For example, password knowledge is necessary for most login procedures. A proof by knowledge can be conducted by a direct demonstration like typing a password. Direct demonstration is not preferable from security point of view since a malicious intended verifier can record or store the information and later use that information to claim the identity of claimant. Indirect demonstration like correctly giving replies to the verifier challenges is the preferred one. Indirect demonstration is used to induce high confidence in the verifier without leaving any clue to how the claimant’s replies are computed. For example, Fiege, Fiet and Shamir propose a zero-knowledge protocol for proof of identity. This protocol allows claimant C to prove to verifier V that C knows how to compute replies to challenges without revealing the replies. These protocols are probably secure [3].

**Proof by possession:** The claimant produce an item that can only be possessed by a principal with the claimed identity, for example, an ID badge. The item has to be safely guarded.

**Proof by Property:** The verifier directly measures certain claimant properties using biometric techniques like retina and fingerprint. The measured property has to be distinguish. No two claimants can have the same value of that property.

Proof by knowledge and proof by possession can be applied to all types of authentication needs in a secure distributed system while, proof by property is generally limited to the authentication of human users by a host equipped with specialized measuring instruments [12].

### 3.3.4 Authentication Flaws

We present below some common flaws in Authentication procedure for distributed system-

**Unconstrained Delegation:** A credential can be delegated when given to person who not only authenticate you but authenticate as you, to somebody else. This is a useful property but also a dangerous one. The username and password are the best example of delegable credentials. In attempt to replace passwords with security tokens, only some of the undesirable delegation properties are addressed. Designer of a system should ask, if a user would be unwilling to give their password to a party who wants to act on their behalf:
Why are they unwilling?

Does my system constrain delegation in a way that addresses these concerns?

Typical reasons a user might be unwilling to share their passwords can include:

Inability to audit actions taken on behalf of a user to the delegated-to party.

Inability to grant access for a limited time and inability to revoke access.

A good protocol will address all these concerns and will not just provide an equivalent of password with another name.

**Unbound Composition of Transport and Message Security:** The composition of necessary properties of an authenticated exchange using both transport and message-level security is the common pattern for modern cryptographic protocols. A transport protocol usually TLS, is used to provide confidentiality, integrity and to authenticate the server to the client. Authentication of the client by the server is provided by an inner rely over the secure transport. This provides convenience, flexibility and performance [7].

In addition to providing authentication and confidentiality, cryptography can serve to bind together parts of a message in a protocol. When message parts are protected independently, an attacker may be able to separate the message parts and recombine them in unintended ways.

**Un-Scoped or Over-Scoped Authority:** The typical identity, authentication and authorization system has some domain of names that can identify principals in the system. The problem of un-scoped or over-scoped authority arises when a party issuing claims is trusted to issue them for parts of the namespace for which it is not authoritative. PKIX, the global PKI used for SSL is the worst offender in this category. In practice, every public CA configured for a platform is trusted to certify every name. There are dozens of public authorities and are all equally trusted, though all are not equally trustworthy. These same CAs are trusted by default to certify non-unique or internal names, such as single label hostnames, IP addresses [6].

**4 Conclusion**

A survey of various checkpointing algorithms shows that a large number of papers have been published. We have considered communication between processes only through message-passing approach and does not include shared memory approach. A majority of algorithms are based upon Chandy-Lamport algorithm by modifying the assumptions made by them. The Table 1 gives comparison of various checkpointing algorithms on some basic parameters. We can simply conclude that the higher the level of abstraction provide by the algorithm, the better is the algorithm to understand and implement. The snapshot algorithms have a large number of applications like checkpointing, detection of stable properties, monitoring, debugging, analyses of distributed computation, etc. All algorithms that don’t rely upon clock synchronization or knowledge of system topology requires \( O(n^2) \) messages. When used in the large systems, the quadratic increase in number of messages may cause large delay. We can reduce this delay if each process keeps a table of the number of messages it has sent to each process in the system and the total number of messages it has received. We can get message complexity in linear time. Storage space used by algorithms vary from one algorithm to another. Some algorithms require large amount of storage space whereas, other algorithms require less amount of storage space.
In the later half of paper, we discussed about various security aspects in distributed systems especially authentication. With the growth in scale of distributed systems, security has become a major concern. For example, security has been strongly advocated as one of the major design constraints in both the Athena and Andrew projects. Most existing distributed systems do not have well-defined security framework and use authentication only for their most critical applications. Various authentication exchanges and approaches for distributed systems have been described in this paper. Authentication flaws are also described in this paper that are need to be addressed so that security of the distributed systems becomes strong and less vulnerable to the attacks.

Acknowledgement

I would like to express my gratitude towards Pushpendra Singh, Assistant Professor @ IIIT-Delhi, for his guidance and constant supervision as well as for providing necessary information regarding the paper and also for his support in completing the paper.

References


