CHAPTER 4

SUBJECTIVE APPROACH TO SERVER SELECTIONS

4.1 INTRODUCTION

RS-ECC based storage was examined in the previous chapter. Based on that a ‘store-retrieve-recover RS coded’ scheme towards enabling a secure and reliable storage was developed. The discussions there established that the original data can be recovered from storage as long as any $k$ non-erroneous data shares are available. Data availability at storages is prone to be constricted when operating the DSS-D across public networks. Significantly, storage failures are hybrid; they include adversaries and the related malicious activities (that corrupt data shares) at storage in addition to other failures and faults. Data availability may be limited not only by server non-participations in accesses but also by incorrect and corrupt (erroneous) shares in the minimal subset $k$ of the $(k + \delta)$ participating servers. It should be noted that data retrievals in effect require reliable servers to be selected from the set of $(k + \delta)$ – malicious and non-malicious – servers. An interesting inquiry of whether storage server selection can be based on behavior / performance at past retrievals was raised in the last chapter. It prompts investigations into a subjective approach towards selections at reconstruction in retrievals.

Subjective server selections necessitate a systematic approach to quantify and manage reliable server information [41] effectively across retrievals. This chapter introduces trust-reputation as a means for judicious server selection. It presents an organized approach to compute and administer ‘server trust’ as a measure of reliability of servers in terms of fragment share submissions, effectively across data retrievals. Section 4.2 discusses role and effect of adversary in a store-retrieve context; a partially trusted operating environment in a
strong adversarial setting is described. Section 4.3 initiates selection of reliable servers on a measure of storage server reliability / trust itself. A trust based selection of the most reliable (minimal) server set for file reconstruction based on servers’ past behavior is systematically developed in Section 4.4. Design of a Trust-Reputation model that establishes ‘server trust’ using Reward and Punishment scheme is detailed. The use of ‘server trust’ in the subjective approach to selection of the minimal subset of $k$ servers for reconstruction is elaborated. Incidentally, this constitutes a key contribution of the thesis. To the best of our knowledge this approach to quantifying and exploiting ‘server trust’ for reconstruction at retrievals is new in DSS-D.

4.2 STORAGE FAILURE MODEL

Unreliability, unpredictability and potentially malicious behavior are expected in distributed storages operating across public networks [7, 42, 48]. Fragment share corruptions at storage servers may occur as a result of hardware errors (DRAM and CPU errors) or adversary taking control of the servers. In addition, the servers may not participate for servicing due to faults at network partitions, crashes and the like. This demands that storage failure models need to be categorically hybrid taking into account both crash and non-crash faults. The reality of non-crash faults in the operating environment calls for an elaboration of the adversary and its effect on storage. Adversary model [42] in distributed storage research has progressed from passive to active [7, 48]. It has further been compounded from being ‘fixed-active’ to being ‘mobile-active’ [7]. It enables categorization and effective modeling of the adversary’s role in the context of storage and retrieval, for both the client servicing and storage layers (Figure 4.1) in DSS-D.

The work presented in the thesis accounts for a partially trusted operating environment in a strong adversarial setting with reasonable computing resources. A partially trusted
operating environment is defined here, by a trusted client servicing layer and an un-trusted storage layer. Interaction between the two involves dispersal and retrieval of client files. The client servicing layer is expected to be likely benign. These servers may delay messages, but will eventually deliver them. On the other hand, the storage layer is expected to be possibly malignant. These servers may exhibit unreliable characteristics that include malicious activities. A strong adversarial setting with reasonable computing resources is assumed here. The maliciousness at servers may be limited to plain fragment share capture. Servers controlled by passive adversary continue to be in the uncorrupted server state. Active adversaries engage in fragment share corruptions. A server in its control makes a transition from uncorrupted to corrupt server state. Servers controlled by active adversary are liable to submit corrupt fragment shares during file retrieval. Clearly, active adversaries at storage disrupt file reconstruction efforts. Further, an active adversary, if mobile, may pose a greater challenge to storage and retrieval. A ‘mobile-active’ adversary does not restrict corruption to a fixed server; instead it progresses through the storage layer corrupting servers, resulting in an eventually degraded servicing. It is important that erroneous fragment shares are dynamically identified & restored to guarantee storage durability and data retrievability with the highest probability.
4.3 RELIABLE SERVER SELECTION

As explained in the previous chapter, file reconstruction is possible from any set of \( k \) fragment shares (out of the \( n \) dispersed); but, it is required that the \( k \) fragment shares be correct and uncorrupted. Data availability at storages is limited in the absence of non-erroneous shares in the minimal subset \( k \) of the participating \( (k + \delta) \) servers. It is therefore important to assure integrity of shares in the minimal subset \( k \). Assurances for integrity therefore become pivotal and customarily require verification of the shares at the \( (k + \delta) \) servers to form the minimal subset \( k \). Pre-computed verification tokens and their variants, cryptography based tools / techniques – cryptographic hash [8, 36, 39], RSA based hash [21], symmetric key hashes [55], homomorphic tokens [64], Proof Of Retrievability, sentinels [7,
24] and the like – are used at large for data integrity checks. Integrity assurances impose a rather heavy dependence on the use of such pre-computed verification tokens.

Decoding at file reconstructions in DSS-D conventionally treats all fragment shares equally. Consequently, this implies that both malicious and non-malicious servers are handled and trusted (or equally not trusted). Such equality is certainly not reasonable in the presence of malicious activities at storage. It clearly brings out the need for a judicious selection of reliable servers, specifically selecting $k$ reliable servers from the set of $(k + \delta)$ malicious and non-malicious participating servers for reconstruction. Conventionally selection procedures base their selection on cryptographic hash or pre-computed token of the fragment shares. Interestingly, trust establishment at storage is an alternative to the use of cryptography based tools / techniques. OceanStore [37] and Farsite [1] use Byzantine protocol to establish trust at storage. On the other hand, it would be more meaningful to base selections on a measure of server reliability / trust itself. System entities can be assigned trust based on the reliable information provided by a Trusted Third Party (TTP); this may not be always feasible [22].

Ruling out cryptographic dependencies, trust establishment at storage and, ‘TTP assigned trust’ for the store-retrieve context entails computing and assigning trust based on direct and/or indirect experiences with the servers. Trust-Reputation schemes, rewarding / penalizing good and bad behavior are effective in computing and assigning entity trust in a system [22, 48]. This calls for a method to explicitly associate a metric that reflects storage server’s reliability / trustworthiness in terms of fragment share submission in retrievals. Such an approach mandates that due weightage be given to storage servers’ past performances in retrievals. Cumulative fragment share submission status of storage servers in the past retrievals forms the logical basis for such a metric – ‘Server Trust Index’ (STI) – introduced in the work presented here. In the existence of non-crash and crash faults at storage, good /
bad behavior of storage servers is quantified in terms of the trust index. This forms an effective basis for formulating a subjective approach to server selection. A subjective based approach to selection here discriminates the servers based on their STI to sift out the $k$ most reliable ones for data reconstruction. Consequently, a minimal set of $k$ reliable fragment shares is facilitated in reconstruction and file is retrieved through ‘trust-based select fragment shares’. The use of ‘server trust’ with its attendant benefits using a reward-punishment method forms the very basis for the work presented here.

4.4 DESIGN OF TRUST-REPUTATION MODEL

As in [63], Trust of any storage server is the belief in its reliability based on a client servicing server’s direct experience with it. Reputation is the belief in storage servers’ reliability based on recommendations received from other client servicing servers. Trust management [41] and Trust-Reputation models may use Analytic expressions, Fuzzy Logic, Bayesian Networks, Social Networks and Bio-inspired algorithms [6, 28, 33, 43, 71]. We propose a method to:

- quantify and manage the trust of storage servers across retrievals using analytical expressions
- facilitate a subjective selection of the most reliable set of $k$ servers at data retrievals

STI (a quantitative measure) that reflects both trust and reputation is used to evaluate the reliability (trust) of a storage server. Here, ‘reliability [4] / trust’ of a storage server implies the extent to which the server is reliable with respect to fragment share submissions in data retrievals. The model facilitates the use of STI for file retrievals, and refreshes the same based on the performance (fragment share submission status) in the current retrieval.
4.4.1 Features

A comprehensive Trust-Reputation model should be context specific, multifaceted and dynamic [63]. The proposed model is comprehensive:

- **Context Specific:** STI of storage servers is evaluated in the context of file store-retrieve servicing.

- **Multifaceted:** Errant servers resulting from adversary activities or hardware errors at storage are taken into consideration. Further, server non-participations due to crash failures are also accounted for.

- **Dynamic:** STI increase / decrease appropriately with retrievals. It may appropriately decay with time in case of steady non-participations and erroneous fragment share submissions.

Behavioral information – past performance in file retrievals – quantified as STI of storage servers is gathered. The servers are ranked based on the STI, the $k$ most reliable servers are selected, and the file is reconstructed in the current retrieval. Subsequently, all the other participant servers in the current retrieval are verified and corrected. Following this, STI is refreshed by appropriately rewarding / punishing the servers relative to their current performance.

4.4.2 Elements

The model primarily incorporates all the elements proposed in [22]. To meet the requirement of ensuring storage correctness, we have extended the framework [22] to include an additional element – ‘Recoverability element’.

(i) **Gathering and Ranking:** Updated STI of every storage server is made available to all the servers in the client servicing layer via a reliable broadcast protocol. The client servicing
server responsible for file retrieval performs the ranking and segregation. Storage server participants are sorted in the order of STI and categorized into three sets – reliable, moderately reliable, and unreliable – based on the quartiles of STI distribution.

(ii) Selection and Transaction: The most favorable set of \( k \) reliable servers is selected from the reliable sets in (i). File reconstruction is performed with the corresponding \( k \) reliable fragment shares.

(iii) Recoverability: Successful reconstruction enables the complete set of \( n \) fragment shares. Verification is performed for the rest of the participants, to identify the servers holding erroneous (incorrect / corrupt) fragment shares. These servers are corrected for their respective fragment shares and restored.

(iv) Reward-Punishment: Erroneous participants are punished and the rest are rewarded. A reward of \( \alpha \) units and penalty of \( \alpha \beta \) units forms the basic units of the scheme.

4.4.3 Reward-Punishment scheme

The analytical expressions for evaluating and updating STI on both participations and non-participations follow.

The proposed scheme gives due consideration for the following recommendations [22] relevant for the store-retrieve context.

A. Prevention of behavioral oscillations along time.

B. Redemption of past malicious servers who have become benign.

C. Prevention of abuse of a good achieved reputation.

D. Assurance of reasonable level of participation for re-entrants.
(i) STI is updated with the basic units of $\alpha$ on reward and $\alpha \beta$ on punishment as:

$$STI_{\text{new}} = S^* + \alpha \text{ on reward, and}$$

$$= S^* - \alpha \beta \text{ on punishment}$$

(4.1)

$S^*$ and $STI_{\text{new}}$ being the current STI value and its updated value respectively.

(ii) Current STI value of a server should be a function of its past STI values. Equal weightage to all past STI values irrespective of their remoteness from the current retrieval may lead to oscillatory behavior. (4.2) alleviates this problem through a progressive reduction of weightage to past STIs, with the use of a first order linear digital filter. This functional relationship is implicitly assumed to be linear here (and the whole of the present work).

**Weighted Average STI:**

$$S^* = \frac{\sum_{i=1}^{n} a^{n-i} b_i S_i}{n_b}$$

(4.2)

$n$, $a^{n-i}$, $S_i$, and $n_b$ being total number of retrievals, weightage for the performance in $i^{th}$ retrieval, server trust index (STI) at $i^{th}$ retrieval, and total number of participations respectively. $b_i$ is 1 if the server participates in $i^{th}$ retrieval and 0 otherwise.

$S^*$ here, conforms to the requirement (A). (4.2) restricts the averaging to only STIs of server participations at retrievals.

(iii) *Eternal punishment of past un-trusted and semi trusted servers with no scope for redemption whatsoever is inappropriate.* (4.3) gives room for redemption of past un-trusted / semi trusted servers (who have become benign) by restricting the averaging in (4.2) to a most recent past. A moving average window length $c + 1$ is used in (4.3).

**Moving Weighted Average STI:**

$$S^* = \frac{\sum_{i=n-c}^{n} a^{n-i} b_i S_i}{n_c}$$

(4.3)

$c + 1$ and $n_c$ are moving average window length and last $c$ retrievals respectively.
$S^*$ here, conforms to the requirements (A) and (B) only. Hitherto non-errant servers are rewarded equally and errant servers are penalized equally.

(iv) *Equal penalty irrespective of the level of reliability / trust may result in classifying a most reliable server, steadily as most reliable, despite it being errant consistently.* These servers may invariably have the advantage of gaining membership into the most favorable set of $k$ reliable servers. The distributed penalty scheme in (4.4) avoids this by giving server penalty proportionate to server reliability – the extent of penalty is directly related to the level of server reliability / trust. Total penalty points is computed and distributed among errant servers such that, a highly reliable errant server (with higher STI) is given relatively more penalty points as opposed to a less reliable errant counterpart (with lower STI) on erroneous fragment share submissions.

The distributed penalty scheme incorporates requirement (C) by using $p$ in place of $\alpha \beta$ units (common / flat units). STI update for participant servers is given as follows.

**(i) Distributed penalty scheme**

$$STI_{\text{new}} = \begin{cases} S^* + \alpha \text{ on reward} \\ S^* - p \text{ on punishment} \end{cases} \quad (4.4)$$

$p$ being the penalty points;

Penalty for $j^{th}$ participant server: \[ p_j = \frac{(S_j - S_0) \beta n_d}{\sum_{i=1}^{n_d} s_i - S_0} \]

$n_d$, $S_0$, and $S_j$ being the total number of errant participants in current retrieval, least STI among $n_d$ participants and last STI of the $j^{th}$ server respectively.

The factor $(S_j - S_0)$ here ensures that the severity of the penalty doled out to a server is commensurate to its reputation. (4.4) takes care of the requirement (C) adequately. The reward and penalty points in (4.4) are for only participant servers.
(v) It is only appropriate to update the STIs of all storage servers after every retrieval and equally not restricting the averaging of STIs to only server participations. $S^*$ in (4.3) is revised for unrestricted averaging of STIs and for non-participant penalty as:

$$S^* = \sum_{i=n-c}^{n} \frac{a^{n-i}S_i}{n_c} \text{ and } p_j = \gamma(\beta n_d)$$

(4.5)

$\gamma$ being a negative number. Thus with a suitable choice of $\gamma$, all non-participants are given common (flat) penalty value.

(vi) ‘Re-entrant STI’ should be devised considering the practical operational aspects of the system, in particular, server replacements (repair and clean up). Re-entrants should assure a reasonable level of participation. Consistent server non-participations and erroneous fragment share submissions, result in STI decay (negative / zero).Decayed STI indicates that the server has to be cleaned / replaced. On re-entry into the system these servers are initialized with an average of reliable storage servers’ STI as:

$$STI_{new} = S^{**} \text{ for re-entrant servers and } S^{**} = \frac{\sum_{i=1}^{n_r} S_i}{n_r}$$

(4.6)

$n_r$ is total number of storage servers in the ‘reliable’ category.

This ensures that replaced servers re-enter the system seamlessly. The reward / punishment scheme in (4.6) includes STI update for re-entrants and satisfies requirement (D) also.

The comprehensive reward-punishment scheme for server STI update developed here is summarized in Figure 4.2. The model developed here is context specific, multifaceted and dynamic. It uses a moving dynamic weighted average with reward and distributed penalty. It takes into consideration the cases of participation, non-participations, and consistent erroneous participations. Flagging of non-participations and erroneous fragment share submissions – STI decay – and the subsequent re-entrant STI are handled. In essence, the model complies with all four recommendations and is comprehensively suitable for store-retrieve context.
Participant and non-participant server STI is updated as:

\[
\text{STI}_{\text{new}} = \begin{cases} 
S^* + \alpha & \text{on reward} \\
S^* - p & \text{on punishment} 
\end{cases}
\]

and \( S^* = \frac{\sum_{i=n-c}^{n} a^{n-i} S_i}{n_c} \)

Penalty for \( j \)th participant server: \( p_j = \frac{(S_j - S_0)\beta n_d}{\sum_{i=1}^{n_d} S_i - S_0} \)

Penalty for \( j \)th non-participant server: \( p_j = \gamma (\beta n_d) \)

Re-entrant server STI is updated as:

\[
\text{STI}_{\text{new}} = S^{**} \quad \text{and} \quad S^{**} = \frac{\sum_{i=1}^{n_r} S_i}{n_r}
\]

Figure 4.2 Reward- Punishment scheme for server STI update

4.4.4 File reconstruction

Selection of the \( k \) storage servers for file reconstruction is essentially using the top \( k \) of the servers in the order of STI – most favorable set of \( k \) reliable servers. The file is reconstructed from the \( k \) fragment share set – ‘trust-based select fragment shares’– obtained from this set of most favorable \( k \) reliable servers. However, this does not imply the correctness of the file or correctness of the \( k \) fragment shares used in reconstruction. A simple yet effective procedure to check for correctness is to append a CRC-based checksum to the file \( F = F \parallel C_\sigma \) prior to the file dispersal. The checksum is recomputed after file reconstruction. Its tally with the appended value in the reconstructed file is the check for file correctness. Conspicuously, it is also the check for the correctness of the \( k \) shares used in reconstruction. Corrupting the
stored file without changing the checksum is of course a possibility. But, for this the adversary has to access all the $n$ file fragments, reconstruct the file, carry out the corruption, and perform dispersal. Undoubtedly all the $n$ storage servers have to be compromised. Obviously this is too tall an order on an adversary, to be feasible. Though straightforward, the check for correctness proposed here fully serves the purpose$^1$. If the resultant file is incorrect, successively other $k$ reliable fragment share sets are to be selected from the rest of the reliable servers in combinations for reconstruction; the same procedure continues until the correct file is reconstructed.

It is important to note that consistent faulty behavior of servers should invariably result in it being classified as one of the most unreliable ones and vice versa. Importantly, any occasional misclassification (false positive) in the most reliable set can be solely due to a compromise / error at the server between the preceding and the current access. On reconstructing the correct file, fragment share verification and correction is seamlessly performed on rest of the participating servers. Assignment of reward and penalty values and STI updation for the participants and non-participants are done conforming to (4.4 and 4.5). The subjective approach significantly paves way for treatment of incorrect storage servers in a singular way, statistical means of reward and punishment of the servers, and selection of $k$ reliable servers. This approach to Trust-Reputation quantifies ‘server trust’ and updates STI to represent the behavior of storage servers in data retrievals in a meaningful manner. The model enables trust based selection of the most reliable, minimal server set for file reconstruction. It is systematically developed to conform to the framework and crucial recommendations [22, 63] demanded of a Trust-Reputation system.

$^1$Appending a CRC bit stream to the file prior to encoding for distribution has been suggested here. It is a small price to be paid for the attendant benefits in terms of a very reliable overall check on file correctness and a conspicuous improvement in the reliability and durability of the DSS-D.
4.5 TRUST-REPUTATION DESIGN – DELIBERATION

A typical data access at storage includes the phases as depicted in Figure 4.3. The approach to server reliability / trust quantification and its updation is summarized in the sequel.

1. Gather Server Trust Index of servers and compute Server Reliability
2. Rank servers based on Server Reliability
3. Select the most favorable set of k reliable servers
4. Perform transaction – File read / write
5. Perform recoverability – Verification and correction of erroneous shares follow File read
6. Decide on reward and punishment and perform Server Trust Index update for all servers

Figure 4.3 Phases in a typical data access at storage

I. Server reliability / trust quantification

We explicitly associate a metric (STI- Server Trust Index) that reflects storage server’s reliability / trustworthiness in terms of fragment share submission in retrievals. Cumulative fragment share submission status of storage servers in the past retrievals forms the logical basis for such a metric – ‘Server Trust Index’ (STI) – introduced in the work presented here. STI (a quantitative measure) that reflects both trust and reputation is used to evaluate the reliability (trust) of a storage server. Here, ‘reliability [4] / trust’ of a storage server implies the extent to which the server is reliable with respect to fragment share submissions in data retrievals. It is an estimate of the extent to which each server can be trusted to supply the correct data share.

Reliability \((S^*)\) is quantified for each server using a moving weighted average scheme in a defined window length – \(l\) with progressive reduction of weights – \(a\).

\(S^*\) of a T2 server at an access is defined as a function of the access performances in the most recent, past accesses as:

\[ S_{i}^{*} = f(I_{i1}, I_{i2}, ..., I_{ij}) \]

where, \(i\) refers to the \(i^{th}\) storage server, \(I_{ij}\) the STI of the server for the \(j^{th}\) access, with \(j\) varying from 1 to \(l\) (\(l^{th}\) access being the most recent) and \(l\) the window length.
A weighted metric restricted to access performances in the most recent past could be one such method enabling a reasonable estimate of the reliability of storage servers. A generalized functional form of the same is:

$$S_i^* = f(a_{i1} I_{i1}, a_{i2} I_{i2}, ..., a_{ij} I_{ij})$$

where, $$a_{ij}$$ is the weight for $$I_{ij}$$ for the $$j^{th}$$ access, with $$j$$ varying from 1 to $$l$$.

II. Updating server reliability (Reward- Punishment scheme)

The servers are ranked based on this estimation ($$S^*$$). To perform an access (transaction), the essential, minimal subset ($$k$$) is selected based on the reliability estimate rank. On successful servicing, each server’s contribution in the access is evaluated in terms of data share correctness; the servers are awarded reward ($$r$$) and penalty ($$p$$) points, followed by an appropriate updation of reliability values. In a reward and punishment method, reliability updation for the $$i^{th}$$ server in an access is as follows:

$$I_i = S_i^* + r_i$$ on reward

$$= S_i^* - p_i$$ on penalty

The expectation from a server is commensurate to its reliability level; this implies that penalty to an erring server should be proportionate to the differential between expected and actual service in the access; this implies penalty of server $$j$$, $$p_j$$ to be a direct function of $$S_{*j} - S_0$$; with $$S_{*0}$$ being least $$S^*$$ among the errant participants and $$S_{*j}$$ being the $$S^*$$ of the $$j^{th}$$ server. The model facilitates the use of STI for file retrievals, and refreshes the same based on the performance (fragment share submission status) in the current retrieval. The pseudo code for server reliability / trust quantification and updation follows:

1. Server reliability quantification

   Let $$S_i^* =$$ server reliability = current_STI

   $$S_i^* = f(a_{i1} I_{i1}, a_{i2} I_{i2}, ..., a_{ij} I_{ij})$$

   where, $$f(a_{i1} I_{i1}, a_{i2} I_{i2}, ..., a_{ij} I_{ij})$$ moving weighted average scheme of server trust indices ($$I$$) in a defined window length – $$l$$ with progressive reduction of weights – $$a$$.

   do for $$n$$ servers
   {
     Compute $$S^* = \frac{\sum_{i=1}^{n-i} a^{n-i}S_i}{n_c}$$ as given in Figure 4.2
   }

   end do
2. Decide penalty

Let transaction = file read / write

if (transaction)
{
    do for n servers
    {
        if ($j^{th}$ server = participant errant server in transaction)
            //penalty to an erring server is proportionate to level of reliability
            
            \[
            \text{Compute } p_j = \frac{(S_j - S_0)\beta_n d}{\sum_{i=1}^{n} S_i - S_0} \text{ as given in Figure 4.2}
            \]
            where, penalty is a direct function of $S^*$ of the $j^{th}$ server-the least $S^*$ among errant participants
        
        else if ($j^{th}$ server = non-participant server in transaction)
            //all non-participants are given common (flat) penalty value
            
            \[
            \text{Compute } p_j = \gamma(\beta n_d) \text{ as given in Figure 4.2}
            \]
    }
}

} end do

} end if

3. Server reliability update

do for n servers
{
    if (! re-entrant server)
    {
        \[
        STI\_new = \left\{ \begin{array}{l}
        S^* + \alpha \text{on reward} \\
        S^* - \alpha \text{on punishment}
        \end{array} \right.
        \]
        \[
        \alpha \text{ being fixed reward}
        \]
    
    else

        //re-entrant servers are initialized with an average of reliable storage servers’ STI

        \[
        STI\_new = \frac{\sum_{i=1}^{n_r} S_i}{n_r} \text{ as in Figure 4.2}
        \]
    } end if
}

} end do
The sequel brings into perspective, approach to trust and its facets in terms of the window of past, reward, and penalty for the trust-reputation model here. A trust-reputation model should take into account several factors in assuring the four (A to D) recommendations for a store-retrieve context. Trust-Reputation is built over a period of time and generally a recommender system looks into the past for a recommendation of trust / reliability in the present. The effectiveness of the recommendation depends on how deep the recommender looks into the past (the window into the past). The inquiry is into how much of the past and relatively remote past should be taken into account? This window into the past must in effect include the immediate past and its past, excluding the remote past. The window length \( c \) in (4.5) reflects the same. The next question is whether discrimination is required in the window or do we give equal importance to all entries in the window? Giving equal weights to the entries irrespective of its remoteness from the present may lead to oscillatory behavior of the system. A progressive reduction of weight (importance) to the past in the window using a first order linear digital filter allows a fair and a good look into the past in building the recommendation of trust / reliability. The weight \( \alpha \) in (4.5) reflects the same. \( S^* \) in (4.5) is a function of the most recent past values. This functional relationship is implicitly assumed to be linear here (and the whole of the present work), this being the simplest representation.

Reward is a common and flat rate of \( \alpha \) units for all non-errant participants. Since system entities are expected to exhibit normal / regular behavior (a storage server is expected normally to store and serve reliable fragment shares), all non-errant participants are treated equally. Any deviation from the expected normal behavior deserves penalty. Generally deviation is inversely related to the expected reliable behavior of system entities; hence, making penalty commensurate to the degree of deviation is more meaningful than maintaining as a mere common flat rate (as it is with reward). In line with this logic, \( \alpha \beta \) unit in the basic scheme is replaced by distributed penalty \( p \) in (4.4) for all errant participants. The
factor \((S_f - S_0)\) in the computation of \(p\) ensures that the severity of the penalty doled out to a server entity is commensurate to its trust-reputation. On the other hand, the non-participants (unavailable) are also given penalty – common penalty value of \(\gamma\) units in (4.5).

The scheme as discussed here is in fact an adaptation of a filter in Digital Signal Processing. Hence, a brief digression into that is in order here. A linear first order model is the simplest representation of a system. Such a discrete system has the transfer function:

\[ h(z) = \frac{a}{1 - az^{-1}}, \]  
where \(a\) represents the pole and \(\alpha\) is the coefficient associated with it. \(h(z)\) represented as the infinite sequence: \(h(z) = \alpha (1 + a z^{-1} + a^2 z^{-2} + a^3 z^{-3} + \ldots )\), is the \(z\)-transform of the impulse response of the system. With a sequence \(\{x[i]\}\) as input to this system, the output \(\{y[i]\}\) can be obtained by convolution; \(\alpha\) is taken as unity here. \(y[i] = \sum_{j=0}^{\infty} x[i - j]a^j\). The effect of \(x[i]\) for past values beyond a desired limit can be ignored by truncating the impulse response suitably. With \(j\) extending to \(l\) we have: \(y[i] = \sum_{j=0}^{l} x[i - j]a^j\). Formation of \(y[0]\) for \(l = 4\) is depicted in Figure 4.4. Subsequent values of \(y[i]\) can be formed by moving the impulse response window successively.

![Figure 4.4 Formation of first order system response](image-url)
The successive values of STIs can be looked upon as a sequence of signal samples (represented as \{x[i]\}). With this the problem of deciding the next STI value based on a set of immediately preceding STI values is analogous to extrapolation based on an available signal sequence. The exponentially decaying weightages for a definite window size corresponds to a linear first order filter type extrapolation as explained above.

4.6 CONCLUSION

The Trust-Reputation model developed here uses Reward-Punishment scheme for server STI update. It is context specific, multifaceted and dynamic. It takes into consideration the cases of participation, non-participations, and consistent erroneous participations. Flagging of non-participations and erroneous fragment share submissions – STI decay – and the subsequent re-entrant STI is handled. In essence, the model complies with all four recommendations and is comprehensively suitable for store-retrieve context. The subjective based approach to selection here discriminates the servers based on their STI to sift out the \(k\) most reliable ones for reconstruction in retrievals. Selection of a minimal set of \(k\) reliable fragment shares is facilitated in reconstruction and file is retrieved through ‘trust-based select fragment shares’.

The fact that storage servers cannot be equally trusted implies that corresponding fragment shares should not be treated equal at reconstructions. We know that conventional RS decoding treats all fragment shares equally and reconstructs the file. Interestingly, the subjective approach to selection presented in this chapter calls for an iterative decoding in the DSS-D. Consequently, conventional RS decoding (HDD) becomes rather an imperceptive choice at decoding. With significantly storage failures being hybrid, optimal iterative decoding is more attractive here. Application of iterative decoding using the trust based approach to optimally identify and use the selected \(k\) fragment shares for reconstruction furthers the scope of the thesis and is portrayed in the next chapter.