HIERARCHICAL FAULT TOLERANT SYSTEM FOR WSNs

WSN is the cooperative engagement of a collection of SNs without the required intervention of any centralized access point or existing infrastructure except the BS. Each node operates as a specialized router and routes are obtained as needed, i.e., on demand with little or no reliance on periodic advertisements. Flat WSNs suffers from poor scalability. Building physically hierarchical WSN is a very promising way to achieve good scalability [207, 36]. To consider this issue, we have developed 3-Tier Monitoring System (3-TMS). In this system we take different characteristics of nodes into account when constructing BNs like node degree and energy resource. The more capable nodes have higher probability of being selected as BN. If WSNs could perform actively, it would be possible to let them maintain a system by themselves. This is quite similar to policing in the real world. Also, it is considered better if fault tolerance mechanisms are available at both the system and application levels, in a mutually complementary style, to make them efficient.

Rest of the chapter is organized as follows. Issues are explored in Section 7.1 System model is presented in Section 7.2. Section 7.3 gives the brief architecture of 3-Tier Monitoring System (3-TMS). Section 7.4 gives a look on protocol which runs on 3-TMS. Analysis of protocol is presented in Section 7.5. Simulation results of 3-TMS is shown Section 7.6, and Discussion about outcomes of 3-TMS model is given in Section 7.7. Finally chapter is summarized in Section 7.8.

7.1 Issues
Existing fault tolerance schemes for WSNs could be centralized or distributed based on whether they are processed at a single or multiple PEs. In the distributed case, a coordinator is needed, which could be a separate process, or be embedded into a daemon process. In the centralized control mechanism, a single process is setup on some PE to monitor the whole system. This monitoring process
constantly dispatches some detective information when conditions are normal. If something goes wrong, it launches a recovery process, to restore the WSN and the crashed processes, from the information that was backed up in secondary storage device of PE(BS/BN/SN). The whole procedure is simple, and causes little network traffic overhead.

However, the bottleneck of the centralized control mechanism is its scalability. If the WSNs span continents, it is unimaginable to setup a monitoring process running at a single place. Such a network suffers from communication delay, making it difficult for the monitoring process to identify the status of processes, if recovery messages are delayed. Thus, performance is degraded. The monitor process suffers from overload, since there may be a very large number of processes running on a large number of PEs (SNs/BSs/BNs). No matter how powerful, it cannot handle the enormous network traffic and requires large memory to maintain information about the whole environment. This mechanism also suffers from a single point of failure. If the PE running the monitoring process crashes, the whole network stops. Although some systems have taken this into account and utilize election algorithms to select a new central monitoring process after such a failure, it is still hard to implement when the system runs over a wide-area heterogeneous network.

7.2 System Model
The existing works, we know that a good fault-tolerant architecture should be able to capture the consistent state of the system, to back up those states in a reliable manner, to detect any failures and start the recovery process. These requirements necessitate the design of completely new fault tolerant system architecture, instead of simply a patch to existing systems. The following undesirable scenarios may occur when MAs are sent from one node/PE to another or BN to another [207-209]:

- When a MA travels from PE (it may be SN/BN/BS) to another, it never reaches its destination, if the destination node has failed or there is a communication link failure, implying that all routes between two node are disabled. Therefore, if a MA at one node wants to travel to another, it will stop
advancing to it and wait until the communication link is enabled again. This is a communication failure.

- When a MA travels from one node to another, it never reaches its destination due to crashes or because it is terminated by some malicious PE (it may be SN/BN/BS) or process (static or mobile). This is MA failure.

- The node, on which a MA resides, crashes or shuts down unexpectedly, due to failures. Many MAs on the node may be in an inactive but waiting state, due to the unavailability of external events. If more MAs migrate to this node, it may run out of memory, energy or both. This is node failure.

7.3 3-Tier Monitoring System (3-TMS)

Distributed control mechanisms are good in terms of scalability and adaptability. Their major weakness is complexity, which often leads to some very complicated implementations. Besides, they also suffer from a high volume of network traffic. Usually, they require $O(n^2)$ communication messages, where $n$ is the number of processes (mobile and static processes both) in the system. The total number of messages increases rapidly as the number of participating PE (SN/BSs/BNs) grows. It is assumed that CPE is equipped with SAP and CHPE & PE are equipped with AMS.

The BN of a subnetwork is called the CHPE and BS of network is called CPE. They are responsible only for receiving requests (MPE: Mobile BSs/MAs), not for executing them. They are an independent PE, assumed to be failure-free. The SAP system installed on them behaves like a gateway and routes mobile processes (MPs) (MAs or mobile device processes) between networks. If a process wants to migrate to another network, CPE/CHPE first checks the status of the PE (BS/BN) of that network. If PE is active, it forwards the MAs, otherwise it (CPE/CHPE) keeps it with itself. PE at the route in network is being actual incharge to execute MAs. And handoff processes required for mobile devices to be executed at CHPE/CPE but executable mobile codes of these devices only executed by route PEs.

Each BN is integrated with MA which collects data from the sensing nodes (root PEs). MA aggregate data and via BNs in the route reaches to BS. MA running on BN may be cloned whenever required for speeding up collection of sensed data. Clones may works on BN as well on sensing nodes their role varies
based on requirement of application. A BN maintains the following information about a WSN.

- Topology map depicting node connectivity and reachability of the network.
- Residual energy map showing battery level of nodes in the network.
- Sensing coverage area map describing the area covered by sensor elements.
- Communication coverage area map presenting communication range of nodes in a network.
- Audit map describing the security status of SNs in a network, whether nodes have been attacked.

Figure 7.1: 3-Tier Monitoring System

In order to maintain scalability and also avoid single points of failure, we have designed and implemented a new framework to handle fault tolerance in WSN
systems, called **3-Tier Monitoring System (3-TMS)** is shown in Figure 7.1. The model works at major three layers, providing fault-tolerance at the root level PEs, between BNs on a subnetwork and between different networks in a global system (between BS-BS), respectively. Layer 3 works at BS and Layer 2 works at BN. Further CHPEs are also able to support to each other for recovering from the failure. In this case CHPE takes help of others if required. Depending on where the fault has occurred in the system, the corresponding layer performs recovery. This optimizes the network traffic, reduces unnecessary communication delays and provides for fast recovery. Thus, we basically follow a centralized control mechanism, with aided WSN and local monitoring schemes.

At the highest level running SAP deputes AMS to work like the Daemon and is called Primary Daemon AMS (PDA) which is installed on the BS (CPE). It controls and monitors processes roaming in a global system of networks, i.e., subnetwork. It also monitors the availability of AMSs and PEs in a subnetwork. It can only look after the MAs during their migration from one subnetwork to another. PDA restarts MAs only on hardware (PE) failure by replacing the failed PE with a backup. When processing on the CPE increases greatly, PDA distributes the load among the available CHPEs on the network and only needs to check their status. If the status report is overdue, it verifies whether the CHPE is down or the message is delayed due to the network traffic. Further, if the status report is overdue from root PEs, CHPE verifies whether the root PE is down or the message is delayed due to the network traffic.

At the middle level a Secondary Daemon AMS (SDA) is present, on every CHPE in the subnetwork. CHPE runs the AMS and periodically checks the status of the root PEs and AMSs in the local network. If an AMS or PE is down it reports to PDA. It also monitors the status of MAs migrating within the network and helps to recover them on failures inside the network.

At the lowest level is AMS is called **Daemon AMS (DA)**, which is present on every root PE(node) of the network and works either on the instruction of the PDA running at the CPE and SDA running at CHPE. It checks the status of the AMS and the MAs executing on a PE. When a MA or AMS is found to be down, DA restarts them on the instruction of PDA and SDA.

It is assumed that the DA, PDA and SDA are all fault free, unless there is a hardware failure, which is rare. The atomic model is considered, i.e., one object
per PE. The PDA and SDA function identically but the SDA gets instructions from the PDA to set system level properties like Mean-Time-To-Recover (MTTR) and Mean-Time-To-Failure (MTTF) value, etc. MAs are implemented as multihop with weak migration for the mobile codes and strong migration for mobile devices.

Note: The word daemon has been used to describe the PDA, the SDA and the DA, because they act as servers for MAs/mobile devices and their messages, but are activated only when required, like daemons.

### 7.4 Recovery Protocol

Figure 7.2 illustrates the steps taken by daemon AMS $DA_i$ to detect failure of mobile process (process executed for mobile device registration, deregistration and MA) $A$, when $DA_{i-1}$ fails to receive registration message $M_{\text{reg}}^i$ and deregistration message $M_{\text{dereg}}^i$. The protocol that is based on assumptions made above is as follows.

1. When a process $A$ arrives at $PE_i$ it registers with message $R_{\text{reg}}^i$ and $DA_{i-1}$ receives its charge at $PE_{i-1}$ from the previous nodet’s SDA running at CHPE $DA_{i-2}$. Then $A$ sends message $M_{\text{reg}}^i$ to $DA_{i-1}$.

2. $DA_{i-1}$ waits for the message $M_{\text{reg}}^i$ for a timeout period and if the timeout period is reached, it requests $DA_i$ to find the status of the process $A$.

3. $DA_i$ searches the registration database for the registration information $R_{\text{reg}}^i$ when it receives requests from $DA_{i-1}$.

4. If $R_{\text{reg}}^i$ is found, $DA_i$ retransmits $M_{\text{reg}}^i$ in order to recover the lost message. If $R_{\text{reg}}^i$ is not found, $DA_i$ recovers $A$ in $PE_i$ by asking $DA_{i-1}$ for the checkpointed data, if $PE_{i-1}$ fails to provide the same, $DA_i$ requests for the same from PDA running at CPE. After completion of the recovery procedure $DA_i$ retransmits the message $M_{\text{reg}}^i$.
5. A performs the assigned task(s) at PE, and after completion checkpoints the intermediate states at PE and in CPE. It then registers message $M_{\text{dereg}}^i$ at PE.

6. If the checkpointing action fails, the process takes the help of SDA running at the CHPE, if still it fails to retrieve. Further it takes help of PDA at CPE. If it still fails, it aborts the whole transaction.

7. Before A migrates from PE, it sends $M_{\text{dereg}}^i$ to DA$_{i-1}$. Upon receiving $M_{\text{dereg}}^i$, DA$_{i-1}$ gives charge of A to DA$_i$ at PE.

---

**Figure 7.2:** Scenario when DA$_{i-1}$ fails to receive $M_{\text{reg}}^i$ or $M_{\text{dereg}}^i$ from PE

### 7.5 Analysis of Protocol

In 3-TMS every MA bears a unique identification number $i$. The SDA monitoring the process $i$ has the same process ID number $i$. The system distinguishes processes and SDA by recognizing their types. A process with process ID $i$ has an
route list $I_i$. $I_i$ is a list of PE ($PE_0, \ldots, PE_{m-1}$) where $m$ is the number of PEs in the route. There are $n$ PEs in the system. The $PE_i$ contains a processing unit $PP_i$ and a secondary storage device $S_i$. We assume that there only exist stopping failures. Other failures such as multipart failure [210] do not exist. Failure of a $PE_i$ is defined as follows: processor of $PE_i$, $PP_i$ fails to advance in the computation and storage $S_i$ fails to operate. This implies that the storage $S_i$ does not fail if the $PP_i$ is working. On the other hand, when $PP_i$ fails, then $S_i$ should fail. We further define that when $PE_i$ fails, all processes running inside it will be terminated.

Time is measured in clocks. Every event in the system should last for an integer multiple of clocks. We define time constants for different events in the system as follows:

- The time needed for a process to complete computation in $PE_i$ is $e_i$ clocks.
- The time needed for a process travel to $PE_i$ from $PE_j$ is $a_{ij}$ clocks.
- The time needed for a message travel to $PE_i$ from $PE_j$ is $m_{ij}$ clocks.
- The time needed for a SDA to recover a process in $PE_i$ is $r_{ai}$ clocks.
- The time needed for PDA (CPE) to inspect and recover $PE_i$ is $r_{ai}$ clocks.

where $e_i, a_{ij}, m_{ij}, r_{ai}, r_{ai} \in N$, and $e_i, a_{ij}, m_{ij}, r_{ai}, r_{ai} > 0$.

We define different time periods in the systems as follows:

$T_r$ = The time for PDA to recover a non-functioning PE.

$T_u$ = The timeout of waiting for $M^i_{reg}$.

$T_i$ = The timeout of waiting for $M^i_{dereg}$.

where $T_u, T_i \in N$ and $T_u, T_i \geq 0$.
7.5.1 Assumptions

- We assume that the processors in different PEs have the same computing speed, so that the amount of computation needed for a process at every PE is same.
- For simplicity we assume that the topology of the network is a complete graph. This implies that every process and message can travel to every other PE in the network.
- Further, we assume that the number of clocks needed for a message to travel is unique throughout all the PEs, i.e., \( m_j = m^*, \forall i, j \in \{0,1,2,\ldots,n-1\} \), and \( m^* \in N \).
- The same rule applies to process travel, i.e., \( a_j = a^*, \forall i, j \in \{0,1,2,\ldots,n-1\} \), and \( a^* \in N \).
- The above rule also applies to PE recovery time, i.e., \( r_j = r^*, \forall i \in \{0,1,2,\ldots,n-1\} \), and \( r^* \in N \).
- Moreover, \( r_a = r^*, \forall i \in \{0,1,2,\ldots,n-1\} \), and \( r^* \in N \).
- We also assume that \( m^*, a^*, r_a^* \) and \( r^* \) are upper bounds of the required time.

When a process \( A_i \) arrives at \( PE_i \), where \( i \in \{0,1,2,\ldots,n-1\} \), it registers the message \( R^i_{\text{reg}} \) on \( S_i \). On the next clock, it sends \( M^{i}_{\text{reg}} \) to \( SDA_{i-1} \) in \( PE_{i-1} \). It starts its computation in the next clock. After completion of execution, it registers \( M^{i}_{\text{dereg}} \) on \( S_i \), sends this message to \( SDA_{i-1} \) and finally migrates to \( PE_{i+1} \) on the next clock if active, otherwise it waits for \( PE_{i+1} \) to become active.

When \( SDA_i \) receives charge of \( A_i \) at \( PE_i \), where \( i \in \{0,1,2,\ldots,n-2\} \), it waits for the message \( M_{\text{reg}}^{i+1} \) for \( T_a \) clocks. If \( M_{\text{reg}}^{i+1} \) does not arrive after \( T_a \), \( SDA_i \) sends a request to \( SDA_{i+1} \) and waits for the next \( T_a \) clocks. The \( SDA_i \) waits for \( M_{\text{dereg}}^i \) from \( SDA_{i+1} \), for \( T_j \) clocks, if \( M_{\text{dereg}}^i \) does not arrive after \( T_j \) clocks, a request is sent to \( SDA_{i+1} \) and \( SDA_i \) waits for the next \( T_j \) clocks.
7.5.2 Liveness Proof

Goal is to prove that the system will not be blocked forever under certain conditions. If the system is blocked forever when inter arrival time of failures of PE is greater than the sum of the process arrival time and process execution time. If the system is blocked forever, at least one of the above time periods ($T_a$ and $T_i$) will reach infinity. The first few steps of our proof are aimed at deriving the lower and upper bounds of the time periods. Given that the route of the process is not infinite, if the upper bounds of all the time periods do not approach infinity, the system should not be blocked forever.

Lemma L.1 $r_s^* \leq T_i \leq nr_s^*$

In the worst case, all PEs are stopped and the PDA starts inspecting and recovering the PEs from $PE_{i+1}$. Hence, the upper bound is $nr_s^*$, and lower bound is trivial, i.e., $r_s^*$. Thus, the requirement of lemma is fulfilled, and $r_s^* \leq T_i \leq nr_s^*$ is true. This lemma default case we assume that if hardware will happen, otherwise this lemma will become $r_s^* \approx T_i \approx r_s^*$, i.e., every PE is monitored by local SDA.

Lemma L.2. We define the lower bounds for various timeouts:

1. $T_a \geq 0$

2. $T_i \geq e^*$

Lower bounds of $T_a$:

Figure 7.3 shows the time space diagram [211] of the system. $T_a$ starts counting the moment $SDA_{i-1}$ receives charge of the process at node $PE_{i-1}$. When $A_{i-1}$ sends $M^{i-1}_{dereg}$, on the next clock, it migrates to $PE_i$, and when $A_i$ arrives at $PE_i$, it sends $M^i_{reg}$ on the next clock. Hence, it takes $a^* + e^* + 2$. On the other hand $SDA_{i-1}$ also takes to $m^* + e^* + 2$ clocks to receive charge from $SDA_{i-2}$. Thus, the relation $T_a \geq 0$ is true.

Lower bounds of $T_i$:
The time between a process executing its task and sending the leave message is $e^*$ (between $PP_6$ and $PP_9$ as shown in Figure 7.4). Thus, relation $T_i \geq e^*$ is also true.

The above lemma gives the number of clocks that SDAs have to wait in the absence of faults. Hence, we can set the timeout periods of the SDA to low, because faults are rare events.

**Definition D.1.** Let $\tau$ be the inter-arrival time of failures of $PE_i$, 
\[ \forall i \in \{0,1,2,...,n-1\} \text{ and } \tau \in N. \]

**Definition D.2.** $f_i$ is the failure sequence of $PE_i$ in the system, since a system can fail $\infty$ time, i.e., $f_i \forall \in \{0,1,2,...,\infty\}$, and $m \in Z^+ \cup \{0\}$. Let $S_f$ be a $1xm$ vector. Thus,
\[ S_f = \{f_0,f_1,...,f_{m-1}) \colon f_i \in \{0,1,2,...,\infty\}, i \in \{0,1,2,...,n-1\}, 0 \leq i \leq m-1, m \in Z^+ \cup \{0\}\} \]

$S_f$ defines a failure sequence with inter-arrival time $\tau$. It implies, without loss of generality, $S_{f_1}$ fails first, and then after $\tau$ clocks, $S_{f_{i+1}}$ fails.

**Lemma L.3.** $a^* + e^* \leq \tau \leq \infty$, if the system is not blocked forever.

It is trivial that the upper bound of $\tau$ is $\infty$, i.e., no failure. To prove the lower bound of $\tau$, we require to compute

(i) The minimum clocks for $A_i$ migrating to $PE_{i+1}$, and

(ii) The minimum clocks for $SDA_i$ to transfer charge of $A_i$ to $SDA_{i+1}$. We assume that there exists a failure sequence $S' = (i,i,...)$, where $\|S'_i\| = \infty$, i.e., all failures happen only in $PE_i$.

If $S'_i (i,i,...)$ then there must be a moment of time that $A_{i-1}$ is waiting for the recovery of $PE_i$, and $PE_i$ is just recovered. During $A_{i-1}$ is migrating to $PE_i$, there
should be no failure happens otherwise the process will be lost. The migration of $A_i$ takes $a^*$ clocks. Also the execution takes $e^*$ clocks. Hence, $\tau \geq a^* + e^*$.

$SDA_{i-1}$ takes $m^*$ clocks to transfer charge of process to $SDA_i$. It takes another $\min(T_a)+\min(T_i)$ clocks for transferring charge from $SDA_i$ to $SDA_{i+1}$.

$\therefore T_a \geq 0$ and $T_i \geq e^*$

$\therefore \tau \geq m^* + e^*$

Thus, we can conclude that $\tau \geq m^* + e^*$ and $\tau \geq a^* + e^*$. Hence, $a^* + e^* \geq m^* + e^*$.

\[ \text{Figure 7.3: Minimum time of } T_a \text{ and } T_i \]
**Corollary C.1.** It is impossible for $A$ to complete its route if $\tau < a^* + e^*$.

This corollary follows from Lemma L.3.

**Assertion A.1.** $r_a < e^*$

Assertion guarantees that the time of the process recovery should be shorter than the process execution. The time needed to have recovered is the message transfer time plus the process recovery time, i.e., $m^* + r_a^*$. We do not desire to have an incomplete recovery, hence $m^* + r_a^* < \tau$ must hold. If $r_a^* = e^*$, there is chance of an incomplete recovery by Lemma L.3. Therefore, it would be nice to have $r_a < e^*$. It is also a reasonable assertion because the process recovery should not be as time consuming as the process execution.

**Lemma L.4.** $0 \leq T_a \leq nr_i^* + m^* + r_a^*$

We use induction to proof the upper bound of $T_a$. Let $T^\text{max}_{a(k)}$ be the upper bound of $T_a$ where $\|S_j\| = k$.

- If $k = 0$, from Lemma L.2, it is trivial that $T^\text{max}_{a(0)} = 0$.
- If $k = 1$,

Suppose that a failure strikes $PE_i$ at the clock that $SDA_{i-1}$ receives charge of the process $A_i$ at the host $PE_{i-1}$, and at the same clock, the timer of $T_a$ will start counting.

$: T_a \geq 0$, we choose the time that $SDA_{i-1}$ should wait be 0.

$: A$ failure happens at $PE_i$, $T_a$, will be reached momentarily.

However, $SDA_{i-1}$ is not required to wait for $T_r$ clocks. Instead, it should be $T_r - m^*$ clocks. Hence $T^\text{max}_{a(1)}$ will be sum of $T_r - m^*$, charge transfer time ($m^*$), the process recovery time, and the message traveling time.

$: T^\text{max}_{a(1)} = (T_r - m^*) + m^* + r_a^* + m^*$, i.e., $T_r + m^* + r_a^*$

- If $k = k'$
Let the failure sequence be $S'_f = (i,i,\ldots,i)$, where $\|S'_f\| = k$. After the first failure is recovered, the time when the next failure happens is $\tau$.

However, the time when SDA, at $PE_i$, recovers $A_i$ is $m^* + r_a^*$.

\[ \therefore \text{if the second and further failures can affect the upper bound of } T_{u(k)}^{\max}, \Rightarrow \tau \leq m^* + r_a^*. \]

\[ \therefore a^* + e^* \leq \tau \Rightarrow e^* \leq r_a^* \Rightarrow \text{Contradiction with Assertion A.1} \]

\[ \therefore \text{we can conclude that only the first failure can affect } T_{u(k)}^{\max} \]

\[ \therefore 0 \leq T_{u(k)}^{\max} \leq T_r + m^* + r_a^* \]

\[ \leq nr_s^* + m^* + r_a^* \]

\[ \Box \]

**Lemma L.5.** $e^* \leq T_r < k(nr_s^* + m^* + r_a^*) + (k-1)e^* + 2m^*$

Where $k$ is the number of failures, $k \in N$, and $a^* + e^* \leq m^* + e^* + r_a^*$

We also use induction to proof the upper bound of $T_i$.

Let $T_{u(k)}^{\max}$ be upper bound of $T_i$ where, $\|S'_f\| = k$.

- If $k = 0$ from Lemma L.2, it is trivial that $T_{u(0)}^{\max} = e^*$
- If $k = 1$

Suppose that a failure strikes $PE_i$ after $A_i$ sends out $M_{reg}^i$ and before $A_i$ sends out $M_{dereg}^i$. We have two cases here, either the computation has finished or it has not finished. Since we only have one failure, we can treat these two cases as one. Since we are estimating the upper bound, we assume that the failure happens when the computation is about to be finished, i.e., at least one clock is remaining. The moment that $PE_i$ crushes, $SDA_{r-1}$ is waiting for $M_{dereg}^i$ is $e^* - m^* - 1$ clocks. So that $PE_i$ is recovered after $T_r - (e^* - m^*) - 1$ from the viewpoint of $SDS_{r-1}$. Thus, $T_{u(1)}^{\max} = T_r - (e^* - m^*) - 1 + m^* + r_a^* + e^* + m^*$
\[ T_r + m^* + r_a^* + 2m^* - 1 < nr_a^* + m^* + r_a^* + 2m^* \]

- If \( k = k' \)

(i) Let the failure sequence be \( S'_f = (i, i, ..., i) \), where \( \|S'_f\| = k' \).

After the first failure is recovered, the time when the next failure happens is \( \tau \). The time when \( SDA_i \) at \( PE_i \) recovers \( A_i \), and the recovered \( A_i \) sends \( M_{dreg}^i \) is \( m^* + r_a^* + e^* \). So that if the second and further failures can affect \( T_{a(k)}^{\text{max}} \) then the relation \( \tau < m^* + r_a^* + e^* \) must hold. Hence,
\[
a^* + e^* \leq \tau < m^* + r_a^* + e^*, \text{ i.e., } r_a^* > 0, \text{ which is always true.}
\]

If \( a^* + e^* \leq \tau < m^* + r_a^* + e^* \), the failure sequence \( S'_f \) will always prohibit the computation from advancing. Thus,
\[
T_{a(k)}^{\text{max}} = T_r - (e^* - m^*) - 1 + (m^* + r_a^* + e^* - 1) + (T_r + m^* + r_a^* + e^* - 1) + ... + m^* = k'(T_r + m^* + r_a^* + e^* - 1) + 2m^* - e^* < k'(m^* + m^* + r_a^*) + (k' - 1)e^* + 2m^*
\]

(ii) Using similar argument in Lemma L.4, if we have a different failure sequence, the estimation of the upper bound of \( T_i \) is still the same.

After defining and proving several assertions, definitions and lemmas, we have enough knowledge to prove the live-ness of the system.

**Theorem T.1.** The system is blocked iff \( S_f = (i, i, ..., i) \) and \( a^* + e^* \leq \tau < m^* + r_a^* + e^* \), where \( \|S_f\| = \infty \), and \( i \in (0, 1, 2, ..., n - 1) \).

If the system is blocked then the timeout period must reaches to \( \infty \). From Lemma L.1, L.4 and L.5, only upper bond of \( T_i \) is proportionally increasing with the number of failures. From proof of Lemma L.5, all the consecutive failures must be happening on the same PE with the maximum inter-arrival time \( m^* + r_a^* + e^* \). Moreover, as \( T_i \to \infty, k \to \infty \).
\[ \therefore S_f = (i, i, \ldots) \quad \text{and} \quad a^* + e^* \leq \tau < m^* + r_a^* + e^* \quad \text{where} \quad \|S_f\| = \infty, \quad \text{and} \quad i \in (0, 1, \ldots, n - 1). \]

From Lemma L.5, \( k \to \infty \Rightarrow T_i \to \infty \therefore T_i \to \infty \Rightarrow A_i \) never finishes computation in \( PE_i \) as infinite failures are happening on \( PE_i \). Hence, system will be blocked.

Theorem T.1 states that the system can still be blocked conditioning on the inter-arrival time of failures of a PE. We can estimate the probability that the conditions will happen as follow.

**Definition D.3.** Let \( N_i(t) \) be a counting process such that, at time \( t \), there are \( N_i \) failures happened in \( PE_i \). Let \( T_{(k)} \) denotes the elapsed time between the \((k - 1)^{th}\) and the \( k^{th}\) failure at \( PE_i \). We let the failure inter-arrival time distribution be an exponential distribution. Thus,

\[
P\left(T_{(k)} > t \mid T_{(k-1)} = s\right) = \begin{cases} 1 - e^{-\lambda_i t} & \text{if } t > a^* + e^* \\ 0 & \text{otherwise} \end{cases}
\]

where \( \lambda_i \) is the mean.

**Figure 7.4:** PE Failure Inter-arrival Time Distribution

Definition D.3 states that the PE failure inter-arrival distribution is a conditional exponential distribution as shown in Figure 7.4, if the time is less than \( a^* + e^* \), the probability is zero. Otherwise, the probability distribution is
exponential. This follows from Lemma L.3, since this lemma states that 
\( \tau > a^* + e^* \) in order that the system will not be blocked forever. Hence,

\[
P(\tau^* + e^* \leq \tau < m^* + e^* + r^*_a) = P(T_{k(i)} > m^* + e^* + r^*_a | T_{k-\alpha(i)} = s) - P(T_{k(i)} > a^* + e^* | T_{k-\alpha(i)} = s) 
\leq e^{-\lambda(u^* + e^*)}(1 - e^{-\lambda T_{k(i)}}),
\]

using Lemma L.2.

7.6 Implementation and Performance Study

Faults are simulated by anti DA programs installed on every BN and an anti PDA server program installed at the BS. On the request of the anti PDA program, the anti DA program randomly kills agents. An anti BN program running on a node similarly kills the SNs/BNs. For the simulation we have used basic parameters shown in Table 7.1. To study the performance, we conducted several experiments.

<table>
<thead>
<tr>
<th>Basic specification</th>
<th>500m±500m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network size</td>
<td>Randomized</td>
</tr>
<tr>
<td>Topology mode</td>
<td></td>
</tr>
<tr>
<td>Total number of SNs</td>
<td>1500</td>
</tr>
<tr>
<td>Data rate MAC layer</td>
<td>1Mbps</td>
</tr>
<tr>
<td>Transmission range of SN</td>
<td>60m</td>
</tr>
<tr>
<td>Sensed data packet interval</td>
<td>1 s</td>
</tr>
<tr>
<td>Numbers of BN</td>
<td>Assumed as per naming scheme presented in Chapter 6.</td>
</tr>
</tbody>
</table>

Table 7.1: Simulation setting

We have compared the 3-TMS model with WFBA[205], and DFDNM [206] simulated on MATLAB to compare their round-trip time and reliability. Reliability is measured as the fraction of agents completing their scheduled round-trips in a network of SNs, BNs, and BS. The anti SDA program randomly kills SNs with exponentially distributed mean time MTTF = 2000 ms. When SDA finds a failed SNs/BNs in the network, it restarts BNs within an exponentially distributed mean time MTTR = 1000 ms.

Figure 7.5 compares execution with and without node (SN/BN) failure detection. It illustrates how reliability is improved under failure detection and recovery with any number of node failures occurring in the network. As the number of nodes increases, the drop in percentage of successful agents under
failure detection and recovery is much slower than without fault recovery. With 50 nodes in the agent itinerary, the drop is between 98 and 52% for WFBA, 98 to 59% for DFDNM and 100 to 82% for 3-TMS with fault detection. This figure is below 20% for all these schemes, without fault detection. The increasing slope implies that the advantage of the fault-tolerant implementation becomes more significant as the number of nodes increases, since the number of successful round-trip-travels decreases progressively. This is reasonable, since, as the chance of waiting for the recovery of a failed node increases, the probability of agent loss also increases for WFBA and DFDNM Model. But in 3-TMS, if a node fails, the agent visits the next node in the itinerary and retries the failed node later. Thus, the probability of losing an agent does not increase rapidly with increase in number of nodes. It is assumed that a node is not failing due to depletion of energy.

![Graph showing percentage itinerary completed vs number of nodes for WFBA, DFDNM, and 3-TMS.](image)

**Figure 7.5:** WFBA and DFDNM vs. 3-TMS for with Fault Tolerant node (SN/BN/BS) and agent

**WF- With Fault**

It was seen that even with node failure detection and recovery, the system still suffers from agent and information loss. Hence, to increase the percentage of successful round-trip travel by an agent, SDA and DA were implemented. We experimented with an itinerary of up to 50 nodes, the results of which are shown in Figure 7.6. It can be seen that there is an improvement in the percentage of successful round-trip travel for all three models. The 3-TMS recovers failed
agents, i.e., it has 83-98% recovery, compared to about 64-95% for WFBA and 46-90% for DFDNM model.

Whenever an agent is recovered, it may encounter another failure. This generates many extra agents, called runway agents, which are needed to search the earlier ones. The number of such extra agents increases as the itinerary becomes longer. Consequently the complexity of the system increases. In 3-TMS, no runway agent is generated and an agent is never blocked on a BN, when both the following conditions hold:

(a) if the BS functions correctly, and

(b) the sum of the agent arrival & execution time is less than an BN failure inter arrival time.

![Figure 7.6: WFBA and DFDNM vs. 3-TMS for with Fault Tolerant node (SN/BN/BS)](image)

The key difference between the protocol suggested in WFBA[205], and DFDNM [206] and 3-TMS all depend on an unreliable broadcast. WFBA protocol broadcasts a lot of redundant messages which uses most of the network bandwidth. This scheme also uses P2P when number of nodes in the network is being less. 3-TMS message exchange mechanism, on the other hand, is a P2P one, so we can save a lot of redundant messages and same is used in DFDNM [206]. Moreover, 3-TMS handles hardware failures, as the PDA running at the BS finds the list of agents running at the failed nodes and transfers them to a new node.
Another major feature of 3-TMS is that no runway agents are created for recovery. Further, the fault tolerant schemes suggested in the literature are application dependent, while 3-TMS is application independent and can be used for existing WSN MASs also.

7.7 Results and Discussion
We have analyzed the developed framework by implementing it on SAP and comparing its utility and performance with existing fault tolerant schemes for mobile devices/MAs. We found that presented protocol produce less computing and communication overhead, because it is less complex and very few messages are sent over the network. It supports failure detection and recovery of agents on slow (WSN, Mobile Ad hoc Network) and open networks like the Internet and also facilitates offline MA/mobile computing. The cost of a fault-tolerant system increases when the level of fault-tolerance is increased, but for the same degree of fault tolerance, 3-TMS is more cost effective. One limitation of 3-TMS framework is that BS/BN failure blocks MAs in a network, but the occurrence of a BS/BN failure is a rare event.

7.8 Summary
In this chapter, we have presented a scalable framework called 3-Tier Monitoring System (3-TMS), to handle fault tolerance in WSN. 3-TMS provides fault tolerant execution of processes (mobile device processes/MAs) which is based on a realistic view of the current status of MA based computing. It covers all possible faults that might invalidate reliable process execution, migration and communication. To minimize the overhead of fault management, 3-TMS distributed the fault diagnostic scheme into three layers. In the next chapter a case study of SAP is presented.