FRODO DT-Spin Models

V.Sundramoorthy[1], C.Tan[2], P.H.Hartel[1], J.I.den Hartog[1], and J.Scholten[1]

[1]University of Twente, Enschede, The Netherlands
{vasughi,pieter,hartogji,scholten}@cs.utwente.nl
[2]Massachusetts Institute of Technology (MIT), Cambridge, USA
c.tan@mit.edu

June 19, 2006

1 Introduction

In the pursuit of a ubiquitous world, network administration, as we know it will change. Self-configuring systems will replace user configuration, where devices discover their environment, detect and adapt to topology changes, establish communication with each other and share services. Several state of the art systems have been developed towards achieving this objective, including Jini [9] by Sun Microsystems, Universal Plug and Play (UPnP) [8] by Microsoft and Service Location Protocol [1, 6] by the IETF.

In general, service discovery protocols adhere to certain fundamental attributes. To start with, there are two types of entities in the system: *User* and *Manager*. A Manager is a service provider, which has a set of services. Each service is represented as a *Service Description (SD)*, which describes the service in terms of: (1) device type (e.g. printer), (2) service type (e.g. color printing) and (3) attribute list (e.g. location, paper size). A User is an entity that has a set of requirements for the services it needs.

There are two types of service discovery architectures: *registry-based* (e.g. Jini) and *peer-to-peer* (e.g. UPnP). A registry-based architecture has a third entity, called the *Registry*. A Manager *registers* its services at a Registry, and Users discover the services through unicast *queries* to the Registry. In the peer-to-peer architecture there are no Registries, and Users discover Managers through broadcast or multicast queries. The registry-based architecture reduces network traffic and makes a network more manageable by allowing Registries to keep track of arriving and departing services. The peer-to-peer architecture avoids single point of failure problems, as may exist in the registry-based architecture, but increases network traffic. A combination of these two architectures can be implemented to allow the protocol to be more resilient against failure on the registry, and reduce network traffic (e.g. SLP, FRODO). However, unlike existing protocols, FRODO [11] implements resource-awareness. The service discovery tasks are partitioned according to resource constraints, where a resource-lean node depends on more powerful neighbors to complement its discovery tasks. From this point onwards, we focus on Registry-based protocols because the peer-to-peer architecture is a simplification of the registry-based architecture, and can be derived by removing Registry-related factors from this work.

Service discovery protocols perform garbage collection of defunct services. A popular mechanism for garbage collection is *leasing*. The Manager refreshes a "lease" periodically to notify the Registries of its continued existence. The services of Managers that have left the network can be purged automatically if the lease is not renewed.

To ensure subscribed Users remain consistent with a Manager whenever its service changes, most service discovery protocols implement update functionalities. The change can be reflected directly in the SD, or in an evented variable that gives the status of the service.

Section 2 describes related work. Section 3 classifies service discovery mechanisms into four major functions. Section 4 describes the service discovery environment. Section 5 gives the 7 Service Discovery Principles. Section 6

provides a brief description of FRODO. Section 7 describes the modeling and verification methodology of FRODO using DT-Spin. Section 8 analyzes the results of verification. The last section concludes. We also include an Appendix that contains the FRODO models, built in DT-Spin.

2 Related Work

The only other work done in this area is the *Service Guarantees* from NIST in [3], which proposes general guarantees that service discovery protocols should strive to satisfy. Our Service Discovery Principles refine these guarantees and moreover, we provide a methodology to proof using model-checking techniques, that a protocol satisfies these principles.

3 Service Discovery Functions

We state the main objectives of service discovery as:

O1: *Discover services that match requirements*

O2: *Detect changes in service availability and attributes*

Toward accomplishing these objectives, we classify service discovery tasks into four main *functions*; Configuration Discovery, Service Registration, SD Discovery and Configuration Update. The term "configuration" refers to the entities in the system: Manager, User and Registry. The Configuration Discovery and Service Registration functions exist to facilitate service discovery. O1 is accomplished by the SD Discovery function, while O2 is accomplished by the Configuration Update function. Each of the four functions can be accomplished using several different *methods*. We use italics to indicate the methods:

1. **Configuration Discovery** - This function allows Registries to be setup, and identities of entities (e.g. Registries or cluster members) in the system to be discovered. There are two sub-functions of Configuration Discovery:

   (a) Registry auto-configuration - Allows the system to configure one or more Registries automatically through (a) *Registry election* algorithms, or (b) *Registry reproduction*, where a parent Registry spawns a child Registry. The Registry election or reproduction is done based on some criteria such as resource superiority, load threshold, service type or location. Registry auto-configuration is done on the fly, without supervision.

   (b) Entity discovery - Allows entities in the system to discover a Registry or cluster through (a) *active discovery*, where nodes initiate the discovery by sending announcements, or (b) *passive discovery*, where nodes discover the required entities by listening for announcements. In some systems, discovery via active and passive methods is integrated with the underlying routing protocol to optimize bandwidth utilization.

2. **Service Registration** - This function allows Managers to register their services at a Registry. Registration methods include (a) *unsolicited registration*, where nodes request the Registry to register their services and (b) *solicited registration*, where Registries request new nodes to register. The Registry keeps a cache of available SDs, and updates them according to requests from the Managers.

3. **SD Discovery** - This function allows Users to obtain SDs that satisfy their set of requirements. Users may cache the discovered SDs to reduce access time to the service, and reduce bandwidth utilization by avoiding multiple queries. There are two sub-functions in SD Discovery:

(a) Query - This is a pull-based model where Users initiate (a) *unicast query* to a Registry, or (b) *multicast query*. The query specifies the requirements of the User. The Registry or Manager that holds the matching SD replies to the query.

(b) Service notification - This is a push-based model, where Users receive (a) *unicast notification of new services* by the Registry, or (d) *multicast service advertisements* by Managers.

4. **Configuration Update** - This function monitors the node and service availability, and changes to the service attributes. There are two sub-functions in Configuration Update:

   (a) Configuration Purge - Allows detection of disconnected entities through (a) *leasing* and (b) *advertisement time-to-live (TTL)*. In leasing, the Manager requests and maintains a lease with the Registry, and refreshes the lease periodically. The Registry assumes that the Manager who fails to refresh its lease has left the system, and purges its information. With TTL, the User monitors the TTL on the advertisement of a discovered Manager. The User assumes the Manager has left the system if the Manager fails to advertise before its TTL expires, and purges its information.

   (b) Consistency Maintenance - Allows Users and Registries to detect updates on cached SDs. Updates can be propagated using (a) push-based *update notification*, where Users and Registries receive notifications from the Manager, or (b) pull-based *polling for updates* by the User to the Registry or Manager for a fresher SD. (c) In a multiple Registry topology, push-based *update notifications among Registries* can be done to achieve consistency.

4 The Service Discovery Environment

If the environment of a service discovery system causes it to fail, we should like the system to recover from failures. Below we provide a formal description of a system.

1. **System**: A system consists of a set of entities with attributes $(\mathsf{e} \in)\mathsf{E}$, a set of services $(\mathsf{s} \in)\mathsf{S}$. The attributes of the entities, described below, evolve over time $(\mathsf{t} \in)\mathsf{Time}$. Based on the attributes, the entities are divided into three (not necessarily pairwise disjoint) sets $(\mathsf{u} \in)\mathsf{U}$ for Users, $(\mathsf{m} \in)\mathsf{M}$ for Managers, and $(\mathsf{c} \in)\mathsf{C}$ for Registries.

2. **Entity attributes**: Each entity, e has the following attributes, which are subject to change over time:

   - $\mathsf{C}(\mathsf{e}) \subseteq \mathsf{E}$ is the set of Registries discovered by e. An entity is called a Registry, i.e. $\mathsf{e} \in \mathsf{C}$, if it has discovered itself in the role of Registry, $\mathsf{e} \in \mathsf{C}(\mathsf{e})$.
   - $\mathsf{OfferedSD}(\mathsf{e}) \subseteq \mathsf{S}$ is the set of services, offered by e. An entity is called a Manager, i.e. $\mathsf{e} \in \mathsf{M}$, if it offers at least one service, $\mathsf{OfferedSD}(\mathsf{e}) \neq \emptyset$.
   - $\mathsf{Requirement}(\mathsf{e}) \subseteq \mathsf{S}$ is the set of services required by entity e. An entity is called a User, i.e. $\mathsf{e} \in \mathsf{U}$, if it requires at least one service, $\mathsf{Requirement}(\mathsf{e}) \neq \emptyset$.
   - $\mathsf{DiscoveredSD}(\mathsf{e}, \mathsf{e}') \subseteq \mathsf{S}$ is the set of services discovered by e at entity e'. This attribute is typically only used when e is a User and e' is a Manager. We put $\mathsf{DiscoveredSD}(\mathsf{u}) := \bigcup_{\mathsf{m} \in \mathsf{M}} \mathsf{DiscoveredSD}(\mathsf{u}, \mathsf{m})$ for the set of all services discovered by a User, u at any Manager, m.
   - $\mathsf{RegisteredSD}(\mathsf{e}) \subseteq \mathsf{S}$ is the set of registered services in e. This attribute is only used for a Registry.

There are several protocol-dependent parameters used in the Service Discovery Principles:

1. **Connectivity condition**: $\mathsf{Conn}(\mathsf{e}, \mathsf{e}')$: The service discovery protocol is responsible for providing the definition of Connectivity. An example is "if a message is transmitted from either e to e', or vice versa, the message is received." The Connectivity condition is not restricted to valid communication paths. It can also be defined by an application, for example, a security application that detects a malicious entity, and indicates to the service discovery protocol that the node is not available for any further operations.

2. **Disconnect condition**: $\mathsf{DisConn}(\mathsf{e}, \mathsf{e}') : \neg\mathsf{Conn}(\mathsf{e}, \mathsf{e}')$ for "sufficiently long", where "sufficiently long" is a protocol-dependent parameter. The definition of "sufficiently long" includes the Connectivity definition and the period communication did not occur. Examples are the limit of retransmissions or the time period for waiting for acknowledgements.

3. **Global Connectivity**, GC: The condition is satisfied if all entities are connected.
$\mathsf{GC} : \forall \mathsf{e}_1, \mathsf{e}_2 \in \mathsf{E} = \mathsf{Conn}(\mathsf{e}_1, \mathsf{e}_2)$

4. **Number of Registries**, N: the desired number of Registries in the system.

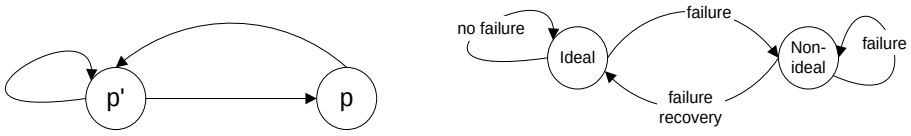
5. **Required Registries or cluster members**, $\mathsf{G}(\mathsf{e})(\subseteq \mathsf{E})$: the Registries or cluster members required by e. For example, an entity may not be interested in knowing all Registries, but only those of a certain type, i.e. those which satisfy its "Registry requirements".

6. **Registry election**, Rank: a function for electing the Registries in the system, where the set C has highestRank if
$\neg(\exists \mathsf{e}' \notin \mathsf{C} : \mathsf{Rank}(\mathsf{e}') > \min\{\mathsf{Rank}(\mathsf{e}) \mid \mathsf{e} \in \mathsf{C}\})$

5 Service Discovery Principles

We use *Linear Temporal Logic (LTL)* [7] to define the Service Discovery Principles. This is essentially positional logic with temporal operators such as $\Box$ (*always*) or $\Diamond$ (*eventually*). As shown in Figure 3.1(a), the recovery of a service discovery system is defined as the *response* pattern, $\Box(p' \rightarrow \Diamond p)$, where p' is the state of Connectivity, Global Connectivity or Disconnect, and p is the response of the service discovery system toward satisfying a Service Discovery Principle.

The Service Discovery Principles define the precise goals of the service discovery functions. Each function must perform correctly, and recover from failures, as shown in Figure 3.1(b). There are two states in a service discovery function. When there are no failures, the function performs correctly and is in the *ideal* state. When failures occur, the function may behave incorrectly, but has to recover from failures and leave the *non-ideal* state when failure ends.



(a) Response pattern for service discovery, $\Box(p' \rightarrow \Diamond p)$

(b) States of a service discovery function,

Figure 1: Service discovery system states. (a) p' is the state of Connectivity, Global Connectivity or Disconnect, and p is the response of the service discovery system to satisfy a Service Discovery Principle. (b) The *ideal* state for a function has no failure, and the function performs correctly. In the *non-ideal* state, the function may perform incorrectly. But when the failure ends, the service discovery function should eventually recover from failures and return to the ideal state.

We need some auxiliary definitions before giving the Service Discovery Principles.

1. The symbols "$\Diamond\subseteq$", "$\Diamond\supseteq$" and "$\Diamond\supset\subset$"
$\mathsf{a} \Diamond\subseteq \mathsf{b}$ holds at time t_0 when a at time t_0 is a subset of b at some future time t_1 where $\mathsf{t}_1 \geq \mathsf{t}_0$ (in LTL, the future includes the present). Similarly, for the symbols "$\Diamond\supseteq$" and "$\Diamond\supset\subset$" where $\mathsf{a} \supset\subset \mathsf{b}$ denotes that the two sets are disjoint ($\mathsf{a} \cap \mathsf{b} = \emptyset$).

2. $\mathsf{ServiceSearch}(\mathsf{u}, \mathsf{c})$ states that a User, u is looking for a specific service from a Registry, c

3. $\mathsf{MatchingSD}(\mathsf{c}, \mathsf{u}) :=$
$\mathsf{RegisteredSD}(\mathsf{c}) \cap \mathsf{Requirement}(\mathsf{u})$
is the set of services registered at the Registry, c which match the requirements of User, u.

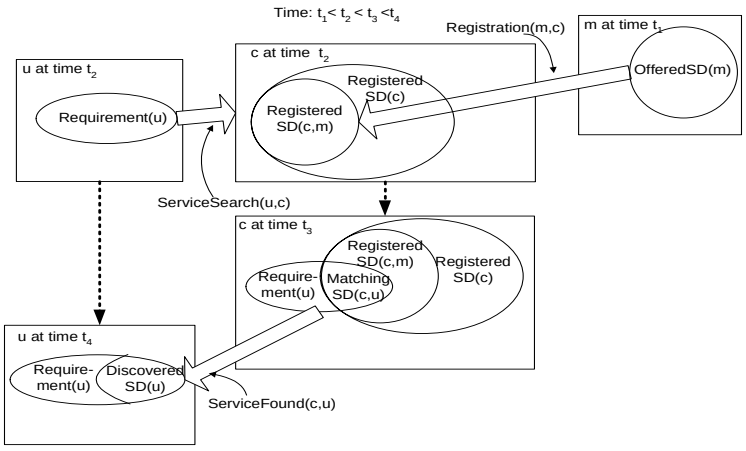


Figure 2: System flow and relations between sets during Registration and Service Discovery. Registration, ServiceSearch and ServiceFound are shown as messages sent between entities. A service is registered by the Manager at time t_1, then discovered by the Registry at t_2, and User searches for the service at or before t_2. The Registry processes the request at t_3, where it finds matching services for the User. The User discovers the service at t_4.

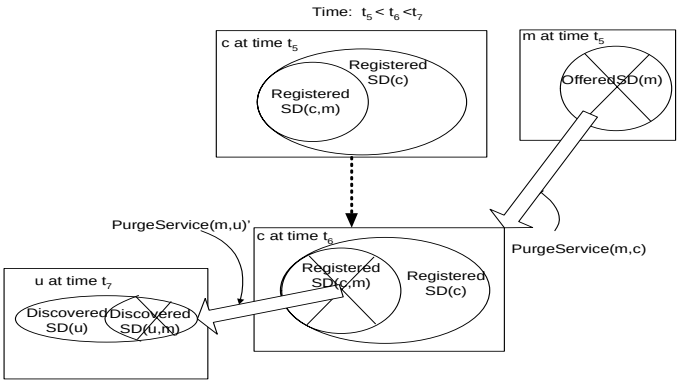


Figure 3: System flow and relations between sets during Configuration Purge. A service is purged by the Manager at t_5, then the Registry is notified at t_6, which then purges the registration. The Registry notifies the User at t_7, and the User purges the service from its DiscoveredSD cache.

4. UpdateSD(m) :=
   states that the OfferedSD(m) of Manager, m has changed.

5. ServiceFound(c, u) :=
   MatchingSD(c, u) $\Diamond \subseteq$ DiscoveredSD(u)
   states that the matching services at a Registry c are discovered by the User u.

6. Registration(m, c) :=
   OfferedSD(m) $\Diamond \subseteq$ RegisteredSD(c)
   states that all services of Manager, m are registered at Registry, c.

7. PurgeService(m, u) :=
   OfferedSD(m) $\Diamond \supset\subset$ DiscoveredSD(u)
   states that the services of Manager, m are eventually purged from the discovered services of User, u.
   PurgeService(m, c) :=
   OfferedSD(m) $\Diamond \supset\subset$ RegisteredSD(c)
   similarly states that the services of Manager m are eventually purged from the registered services of Registry, c.
   If an entity is both a Registry and a User then both properties must hold.

8. Uptodate(u, m) := OfferedSD(m) $\cap$
   Requirement(u) $\Diamond \subseteq$ DiscoveredSD(u, m) $\wedge$
   OfferedSD(m) $\Diamond \supseteq$ DiscoveredSD(u, m)
   states that User u will find new services at Manager m and remove services that are no longer available at m.

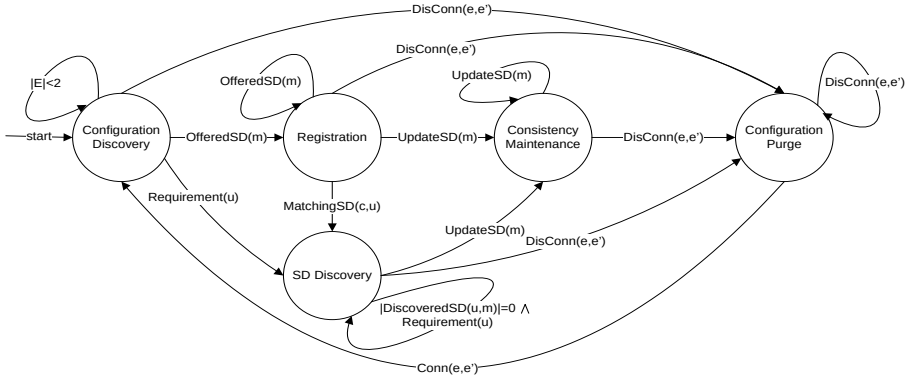


Figure 4: Service discovery system life cycle. When a system consists of more than one entity, Configuration Discovery is performed, followed by Registration or SD Discovery. Registration is triggered when there exists a Manager with OfferedSD(m). SD Discovery is triggered when there exits a User with Requirement(u), or if the Registry has MatchingSD(c, u). SD Discovery is done every time there is a new requirement or as long as the User has not discovered the required service, where $|\text{DiscoveredSD}(u, m)| = 0$. When UpdateSD(m) occurs in the Manager (SD changes), Consistency Maintenance is performed. Configuration Purge occurs every time entities face DisConn(e, e′) due to failures. When failures end, and Conn(e, e′) is restored, the cycle is restarted.

Figures 2 and 3 provide scenarios for a Registry-based system, where the relationship between entities, and the sets are explained, as time progresses. In Figure 2, ServiceSearch is the message providing the requirements of the User to the Registry, while in Figure 3, PurgeService and PurgeService′ are the messages that notify the Registry and User respectively of a defunct service.

We summarize the general life cycle of a service discovery system in Figure 3.4. We show that transitions between functions are triggered by the appearance of services, OfferedSD(m), requirements, Requirement(u), service changes, UpdateSD(m) and conditions DisConn(e, e′) and Conn(e, e′).

With the system and basic properties in place we are ready to give the 8 *Service Discovery Principles*.

(P1) **Registry Setup Principle**
When there is global connectivity, N Registries of the highest rank are selected in the system.

$\Box(\mathsf{GC} \rightarrow \Diamond((|\mathsf{C}| = \mathsf{N}) \wedge \mathsf{highestRank}))$
Assume all entities have the same rank if no ranking function is used.

(P2) **Configuration Discovery Principle**
An entity discovers all available Registries or cluster members in the system that it is interested in.

$\forall \mathsf{e} : \Box(\mathsf{C}(\mathsf{e}) \subseteq \mathsf{G}(\mathsf{e}) \wedge$
$\quad \forall \mathsf{c} \in \mathsf{C} \cap \mathsf{G}(\mathsf{e}) : \mathsf{Conn}(\mathsf{c}, \mathsf{e}) \rightarrow \Diamond \mathsf{c} \in \mathsf{C}(\mathsf{e}))$

(P3) **Registration Principle**
A Manager registers its service description at each Registry it discovers.

$\forall \mathsf{m}, \forall \mathsf{c} \in \mathsf{C}(\mathsf{m}) :$
$\quad \Box(\mathsf{Conn}(\mathsf{m}, \mathsf{c}) \rightarrow \mathsf{Registration}(\mathsf{m}, \mathsf{c}))$

(P4) **SD Discovery Principle**
A User discovers the service descriptions that match its requirements directly from the Manager, or from a Registry.

$\forall \mathsf{u}, \mathsf{e} : \Box(\mathsf{Conn}(\mathsf{e}, \mathsf{u}) \rightarrow \mathsf{ServiceFound}(\mathsf{e}, \mathsf{u}))$

(P5) **Configuration Purge Principle**
A User or Registry purges the services of a Manager that has become disconnected.

$\forall \mathsf{e}, \forall \mathsf{e} \in \mathsf{U} \cup \mathsf{C} :$
$\quad \Box(\mathsf{DisConn}(\mathsf{m}, \mathsf{e}) \rightarrow \mathsf{PurgeService}(\mathsf{m}, \mathsf{e}))$

(P6) **2-Party Consistency Maintenance Principle**
A User remains consistent with a Manager when its services change. This principle applies to consistency maintenance between the User and the Manager, hence the term "2-party".

$\forall \mathsf{m}, \mathsf{u} : \Box(\mathsf{Conn}(\mathsf{u}, \mathsf{m}) \rightarrow \mathsf{Uptodate}(\mathsf{u}, \mathsf{m}))$

(P7) **3-Party Consistency Maintenance Principle**
A User remains consistent with a Manager when its services change, through the Registry. This principle applies to consistency maintenance between the User, the Registry, and the Manager, hence the term "3-party".

$\forall \mathsf{c} \in \mathsf{C}(\mathsf{m}) : \Box((\mathsf{Conn}(\mathsf{c}, \mathsf{m}) \wedge \Diamond \mathsf{Conn}(\mathsf{u}, \mathsf{c})) \rightarrow \mathsf{Uptodate}(\mathsf{u}, \mathsf{m}))$

Although the Service Discovery Principles are tailored for small systems, it is possible to extend the principles to large systems:

- Configuration Discovery Principle: A Registry may need to discover other relevant Registries, based on the type of topology. In a meshed Registry topology, Registries need to discover each other for forwarding queries, while in clustered Registry topology, Registries need to discover the relevant cluster members. The Configuration Discovery Principle can be directly applied in such cases where e is the Registry that has a set of requirement for discovering peer Registries.
- 2-party Consistency Maintenance Principle: A Registry replica requires fresh service information from its parent or peer when there is indication that its cache is inconsistent (e.g. reaches maximum cache misses). In such cases, the Registry requiring the update follows the behavior of the User, u and the Registry providing the update behaves as the Manager, m in the 2-party Consistency Maintenance Principle.

Table 1: Taxonomy of registry-based service discovery systems. FRODO classifies devices into 3C, 3D and 300D, in order of increasing resources.

Functional Area, related Principles and supportive attributes	**Mechanisms**	**SLP**	**Jini**	**FRODO**
Configuration Discovery (*Registry Setup and Registry Discovery Principles*)	Auto-configured Registry	No	No	Yes - through leader election among 300D nodes
	Registry Discovery	Active and lazy discovery	Active and lazy discovery	Active and lazy discovery
Registration (*Registration Principle*)	Registration mechanism	Unsolicited registration	Unsolicited registration	Solicited and unsolicited registration
Service Discovery (*Service Discovery Principle*)	Search mechanism	Unicast (multicast possible)	Unicast	Unicast (multicast possible)
	Notification of new services to Users	No	Yes - unicast notification by Registry	Yes - unicast notification by Registry
Configuration update (*2-party and 3-party Configuration Update Principles*)	Garbage collection	Managers renew lease	Managers renew lease	300D Managers renew lease, Registry polls 3D/3C Managers
	Update information	Service update	Service update	Service update, event notification
	Update mechanism	None - register again	3-party subscription	2-party subscription, 3-party subscription
Supportive attributes	Acknowledgements and retransmissions	TCP-dependent	TCP dependent	Implemented with timers
	Resource awareness	No	No	Yes, device classification
	Service usage	Application layer	Mobile proxy code	Application layer
	Security	Limited, with application driven authetication	None, depends on JDK 1.2 security model	None, future work

6 FRODO

FRODO implements resource-awareness and robustness. For resource-awareness, FRODO classifies nodes as (1) 3C device class - simple devices with restricted resources (e.g. smart dust), (2) 3D device class- medium complex devices (e.g. temperature controller) and (3) 300D device class - powerful devices, controlled by a complex embedded computer, with more than 1MB memory requirement (e.g. set-top boxes). Only a 300D node can become a Registry, through a *leader election* process. The Registry is called the *Central*. The functions of service discovery explained in Section 5 are implemented according to the device classes. The names of the functions remain the same for FRODO, except we classify Configuration Update as part of the wider Configuration Management, to accommodate its robust features. Table 1 provides a summary of the mechanisms that each function implements. The protocol is less dependent than the state-of-the-art on the recovery ability of the lower layers. This allows the protocol to be deployed together with leaner lower layer protocol stacks, with restricted error recovery mechanisms. More details on FRODO can be found in [12, 11]

7 Modeling and Verification of FRODO

We use model checking [5] to verify that FRODO adheres to the Service Discovery Principles. If there are cases where the protocol fails the verification, we identify whether the error lies in the modeling process, or in the design phase. The former requires the model to be corrected (and often our understanding!). However, it is the latter which is

most beneficial, since detecting a design flaw leads towards discovering design solutions that improve and strengthen the protocol, until the Service Discovery Principles are satisfied.

We verify the desired behavioral properties of the protocol through exhaustive enumeration (explicit or implicit) of all the states reachable by the system and the behaviors that navigate through them, given a limited number of system scenarios (but carefully selected based on the service discovery functions in Figure 4). We use the model-checker DT-Spin [2], for this purpose. DT-Spin is an extension of the well-known SPIN tool [5]. DT-Spin is used instead of the original SPIN model checker because we need multiple timeouts that occur at any time, even if other processes are still enabled in the model. Timeouts are essential requirements for the recovery processes of the protocol against failures. The *timeout* variable in standard SPIN cannot be used as it is only true when the system is idle. However, DT-Spin also increases the state space with respect to standard Spin, because time is modeled as ticks, and each tick occupies a state. A reasonable number of ticks is set for each timer, to make verification feasible.

7.1 Modeling Approach

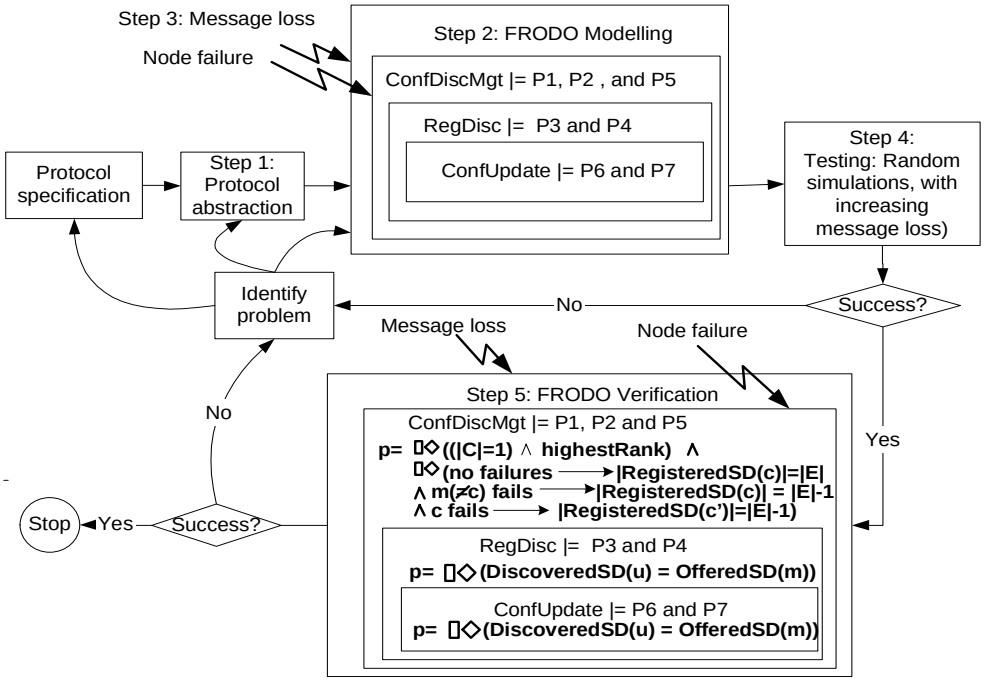


Figure 5: Modeling FRODO. The FRODO Modeling box shows the abstraction link between the 3 modules, where the outer boxes abstract some functions from the inner boxes. Connectivity and Global Connectivity are modeled with/without message loss respectively, and Disconnect is modeled as node failure.

It is impractical to work with monolithic models of complex protocols such as FRODO [10]. In the first place error traces would be too cluttered with irrelevant detail to be able to spot the real problems, and secondly the state space would grow beyond the bounds of what the current tools can cope with. Therefore we split the FRODO protocol in a number of modules, each of which corresponds roughly to one of the four functional areas or sub-functions thereof. Each model is then provided in several versions, including a concrete model with the most detail, some versions that represent worst case behavior, and a number of abstract versions with as little detail as possible. By combining a concrete version of one module with appropriate abstract versions of all others, the system as a whole can be verified, focusing on the behavior of the concrete module. The main challenge of this method is to keep the different versions of each module consistent. Figure 5 shows the approach that we use to model, simulate, and verify FRODO against the Service Discovery Principles.

Step 1: Protocol abstraction. Each assembly of modules builds on a common layer of protocol abstraction. We deliberately abstract irrelevant detail such as message format.

Step 2: Modular decomposition. FRODO consists of three modules: a) *ConfDiscPurge* models Configuration Discovery and Configuration Management. This module is actually broken into four sub-functions as follows: *ConfDiscPurge-1* models the leader election protocol which elects one Registry. *ConfDiscPurge-2* models two worst-case scenarios of the protocol with (i) all nodes claiming to be Registry after a network partitioning, and (ii) all registration information in the Registry being lost because of prolonged communication failure. *ConfDiscPurge-3* models a Backup taking over as the Registry when the existing Registry leaves the system. This model uses node failure, instead of message loss to model network disturbance. *ConfDiscPurge-4* models the Registry handing over to a superior node that enters late into the system. This model is an abstraction of ConfDiscPurge-1, where message loss is already checked, and therefore not included in ConfDiscPurge-4. b) *SrvRegDisc* models both Registration and Service Discovery functions. Configuration Discovery is abstracted away in this model, i.e. a Registry is successfully elected. The model consists of one 300D node which is the Registry. Service discovery is done through the Registry using the directed search mechanism, thus the service cannot be discovered unless registration occurs. The discovery of each type of Manager (3C, 3D and 300D) is modeled separately. c) *ConMain* models the Consistency Maintenance function, which propagates updates to Users through 2-party and 3-party subscription.

Step 3: Message loss. Failure is modeled as message loss. All models except ConfDiscPurge-3 and ConfDiscPurge-4 are simulated and verified with and without message loss. A receiver will continue executing its tasks, unaware of any message loss, which may lead towards a violation of the Service Discovery Principles. We use a counter for every message *type*, which increments whenever a particular type of message is lost. The message may be lost, until a constant *MAX_LOSS* for its type is reached. The following is an example of how a *SrvRegReq* message is sent or lost. The variables *rcvrID, OfferedSD*, and *srcID* are the receiver node's identifier, the service identifier that represents a service and the sender node's identifier respectively.

```
if
:: loss_counter[type]<= MAX_LOSS ->
           lossCounter[type]++
/* transmit message */
:: send!SrvRegReq,OfferedSD,srcID,rcvrID;
fi
```

Message loss increases the number of states because of the additional counter and timing variables and non-deterministic choice. The impact of message loss on the verification is shown in Table 2. In the example, when MAX_LOSS > 1, state space explosion causes the model-checker to halt due to machine memory limitation (we use machines with available memory of 20GB).

Table 2: An example of the impact of message loss on state space. In this example, when MAX_LOSS > 1, the verification halts because of machine memory limitation.

Model	**State vector (bytes)**	**Depth**	**States stored**
ConfDiscPurge-1, exhaustive mode, no message loss	304	37328	38345
ConfDiscPurge-1, supertrace mode, MAX_LOSS=1	308	61993	497,662 billion

Step 4: Testing. We use the simulator tool in DT-Spin to test and debug the models. We test different, random simulation scenarios and increase the MAX_LOSS for every simulation (up to 10 message losses to capture extreme scenarios)

Step 5: Verification. DT-Spin checks correctness claims that are generated from logic formulas expressed in LTL. When a claim is invalid for a model, the tool produces a counter example that explicitly shows how the property was violated. The counter example is a feedback for the simulator tool of DT-Spin to show the execution trail that causes the violation.

7.2 Property Modeling

We now model the desired behavioral properties of FRODO. We use the parameters and notations defined in Section 5to model the properties of FRODO. The interpretations of the protocol-dependent parameters, described in Section 4 are:

1. Connectivity condition: $\mathsf{Conn}(\mathsf{e}, \mathsf{e}')$: *if a message is transmitted from either* e *to* e'*, or vice versa, and the expected acknowledgement arrives, the entities are reachable.*
2. Disconnect condition: $\neg\mathsf{Conn}(\mathsf{e}, \mathsf{e}')$: *for twice the timeout period* (as explained in Section 4).
3. $\mathsf{Rank}(\cdot)$, the function used in Registry election, that returns the highest ranked 300D node as the Registry.
4. $\mathsf{G}(\mathsf{e}) = \mathsf{E}$ the nodes are interested in any Registry as there is only a single Registry in FRODO.
5. $\mathsf{N} = 1$, a single Registry is elected.

We model the response pattern $\Box(\mathsf{p}' \rightarrow \Diamond\mathsf{p})$, by building the property p' directly into the models, so that p is verified as a *recurrence property* [5]. *The recurrence property* $\Box\Diamond\mathsf{p}$ *states that if the state formula,* p *happens to be false at any given point in a run, it is always guaranteed to become true again if the run is continued.* For $\mathsf{p}' =$ Global Connectivity, we verify p in a model without any message loss. For $\mathsf{p}' =$ Connectivity, we verify p in a model with a limit on message loss. For $\mathsf{p}' =$ Disconnect, we verify p in a model with node failure.

The descriptions of each property, p are given below. For the purpose of readability, we left out some technical details. For more details, we recommend the DT-Spin models attached in the Appendix.

$$\mathsf{ConfDiscPurge} \models \mathsf{P1} \wedge \mathsf{P2} \wedge \mathsf{P5}$$

In this property, the number of entities in the system varies according to node failures (e.g. node crashes). We use $|\mathsf{E}|$ to represent the original number of nodes in the system, before a node failure occurs.

For $\mathsf{P1}$, all entities have to agree on the highest ranking node, say c, becoming the single Registry. For $\mathsf{P2}$, all nodes must discover this Registry (and no others). In the verification we check $|\mathsf{RegisteredSD}(\mathsf{c})| = |\mathsf{E}|$, which implies that each node m has registered at c. In this model, the Manager offers exactly one service. The Manager discovers the Registry, and registers. For $\mathsf{P5}$, the Registry purges the service registration of disconnected nodes. Thus, $|\mathsf{RegisteredSD}(\mathsf{c})| = |\mathsf{E}| - 1$. If the failing node is the Registry itself then the Backup c', which is the second highest ranking node, must detect this and take over as the new Registry, resulting in $|\mathsf{RegisteredSD}(\mathsf{c}')| = |\mathsf{E}| - 1$. Thus, for the ConfDiscPurge module, we check the following property (each line is checked separately, in different models)

$$\begin{aligned}
\mathsf{p} := {} & \Box\Diamond(|\mathsf{C}| = 1 \wedge \mathsf{highestRank}) \wedge && /*\,\mathsf{P1}\,*/ \\
& \Box\Diamond(\text{ no failures } \rightarrow |\mathsf{RegisteredSD}(\mathsf{c})| = |\mathsf{E}|) \wedge && /*\,\mathsf{P2}\,*/ \\
& \Box\Diamond(\ \mathsf{m}(\neq \mathsf{c}) \text{ fails } \rightarrow |\mathsf{RegisteredSD}(\mathsf{c})| = |\mathsf{E}| - 1) \wedge && /*\,\mathsf{P5}\,*/ \\
& \Box\Diamond(\ \mathsf{c} \text{ fails } \rightarrow |\mathsf{RegisteredSD}(\mathsf{c}')| = |\mathsf{E}| - 1) && /*\,\mathsf{P5}\,*/
\end{aligned}$$

Together with some basic properties of the behavior of each of the nodes (e.g. discovering only one Registry), it is sufficient to obtain that $\mathsf{P1}$, $\mathsf{P2}$ and $\mathsf{P5}$ all hold.

$$\mathsf{SrvRegDisc} \models \mathsf{P3} \wedge \mathsf{P4}$$

We consider the situation where a User u is interested in the service that is offered by some Manager m in the system, $\mathsf{Requirement(u)} = \mathsf{OfferedSD(m)}$. We verify the property

$\mathsf{p} := \Box\Diamond(\mathsf{DiscoveredSD(u)} = \mathsf{OfferedSD(m)})$

Which gives that services can be found (P4) and also that services are registered (P3) as the User can only discover registered services.

$$\mathsf{ConMain} \models \mathsf{P6} \wedge \mathsf{P7}$$

We consider the situation where the service at m discovered by a User u changes but still satisfies the requirements of the User. To satisfy P6 and P7, the User has to update its discovered services to remain consistent with the Manager, $\mathsf{Uptodate(u, m)}$. This is implied by $\mathsf{DiscoveredSD(u)} = \mathsf{OfferedSD(m)}$. We again check the property

$\mathsf{p} := \Box\Diamond(\mathsf{DiscoveredSD(u)} = \mathsf{OfferedSD(m)})$

where the value of $\mathsf{OfferedSD(m)}$ is changed after a waiting period, during the verification.

8 Verification Results

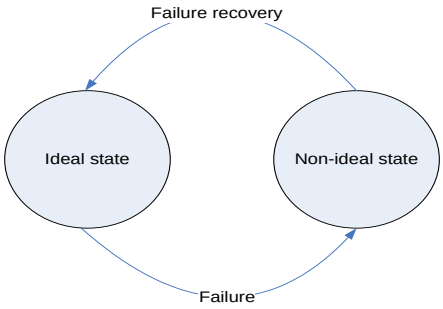


Figure 6: States of a service discovery function. The *ideal* state for a function has no failure, and the function performs correctly. In the *non-ideal* state, the function may perform incorrectly. But when the failure ends, the service discovery function should eventually recover from failures and return to the ideal state.

As shown in Figure 6, a service discovery function oscillates between the ideal state, and the non-ideal state, where communication or node failure is the trigger for the transition. For example, a node can successfully discover the Registry, but fail to register when the registration message is lost. Thus the Registry Setup and Configuration Discovery Principles are satisfied, but the Registration Principle is violated. The system must ensure that the Registration function can return to the ideal state, and satisfy the principle.

We find formally verifying a model of a lossy service discovery system a critical step in the design phase. This is because formal verification reveals design errors that cause the Service Discovery Principles to be violated. These errors may not be detected by simulating FRODO, or by testing the prototype.

We first give the analysis on the failure scenarios, before giving the results of our verification.

Table 3 lists the failure scenarios that cause the Service Discovery Principles to be violated. For each failure scenario, we give the design solution that solves the failure in our FRODO models. The failure scenarios are relevant for any small-scale, unattended system with auto-configured Registries, with unreliable transmission.

We summarize the verification settings, and the results for each module in Table 4. In total, we developed and verified 21 models, out of which, 17 models achieved Exhaustive coverage (100% coverage of all reachable states for

a model), while 4 models had to be run under the Supertrace mode (uses bit state hashing [4], with coverage around 98%) because of state space explosion.

Table 3: Summary of failure scenarios that violate the Service Discovery Principles, and the resulting design solutions. The "Ref" column is used in Table 4 to refer to the design solutions incoporated in the models.

Ref	Failure scenario	Scenario description	Principles violated	Design solution
A	**More than the required N Registries are elected.**	Network partitioning causes multiple Registries to be elected. The system must eventually converge to N Registries.	Registry Setup, Configuration Discovery.	The Registry use *periodic passive discovery* to detect other Registries, and negotiate to satisfy the value of N.
B	**Permanent Registry failure causes the Manager to never be discovered.**	The Registry fails *after* acknowledging the registration, and *before* updating a Backup. When the registration is acknowledged, the Manager keeps renewing its lease, but does not detect that the Registry is no longer available.	Configuration Discovery, Registration, SD Discovery.	The Manager *caches Registry information*, and initiates *unsolicited registration* to unknown Registries.
C	**Temporary Registry failure and cached Registry information causes the Manager to be never discovered.**	The Registry *recovers* from failure, but has *purged* the information on the Manager. The Manager does not attempt to re-register upon receiving the announcement of the Registry, because it has previously cached the Registry information, and assumes its registration is still valid.	Configuration Discovery, Registration, SD Discovery.	The Registry initiates *solicited registration* to detect purged Managers.
D	**Update is unsuccessful, when the Manager or the Registry gives up retransmitting the notification.**	Continuous communication or node failure cause the Manager or the Registry to stop retransmitting the update after the maximum number of retransmission is reached	2-party and 3-party Consistency Maintenance.	The Manager retransmits the update notification until successful (critical update), or retries the update in the future (non-critical update)

Table 4: Verification results. The "success" results are obtained after correcting the failures using the design solutions from Table 3.

Module	**Principle**	**Results**
ConfDiscPurge	Registry Setup, Registry Discovery, Configuration Purge	Success, after implementing design solutions A, B and C
SrvRegDisc	Registration, SD Discovery	Success
ConMain	2-party and 3-party Consistency Maintenance	Success, after implementing design solution D
Verify models during unconnected state	Disconnect check on all principles.	Success. Principles still hold

9 Conclusions

We classify the behavior of service discovery into 4 functional areas, and analyze the mechanisms used in each of these areas. We show that existing registry based service discovery protocols fit into this classification.

The functional requirements of service discovery, needed for correct behavior of applications in the ubiquitous environment, are expressed in 7 Service Discovery Principles. Existing protocols do not satisfy these principles because they rely on the underlying network to provide robustness. In contrast, our own service discovery protocol FRODO does satisfy the principles. We show this by formal modeling and verification.

FRODO is the first service discovery protocol that provides formally verified functional guarantees. The simulation and verification process is beneficial in discovering flaws in the design and identifying essential mechanisms needed to satisfy the Service Discovery Principles.

Future work includes investigating the non-functional aspects of service discovery such as time constraints, by simulation.

10 Acknowledgement

This research is sponsored by the Netherlands Organization for Scientific Research (NWO) under grant number 612.060.111, and by the IBM Equinox program. We thank Christopher Dabrowski and Kevin Mills from the US National Institute of Standards and Technology for their support and contribution to this paper. We also thank NIST for supporting this work through their Foreign Guest Researcher Program.

References

[1] C. Bettstetter and C. Renner. A comparison of service discovery protocols and implementation of the service location protocol. In *Proceedings of 6th EUNICE Open European Summer School: Innovative Internet Applications*, pages 101–108. University of Twente, September 2000.

[2] D. Bosnacki. Implementing discrete time in promela and spin. In *Proceedings of the VIII Conference on Logic and Computer Science, LIRA '97,*, pages 25–32, 1997.

[3] C. Dabrowski, K. Mills, and S. Quirolgico. *A Model-based Analysis of First-Generation Service Discovery Systems*. Special Publication 500-260, National Institute of Standards and Technology, 2005.

[4] G.J.Holzmann. An analysis of bitstate hashing. In *Formal Methods In System Design*, volume 13, pages 287–305. Springer-Verlag, November 1998.

[5] G.J.Holzmann. The model checker spin, primer and reference manual. Addison-Wesley, September 2003.

[6] E. Guttman, C. Perkins, J. C. Veizades, and M. Day. *Service Location Protocol, V.2, RFC-2608*. Internet Engineering Task Force (IETF), December 2003.

[7] M. Huth and M. Ryan. Logic in computer science: Modelling and reasoning about systems. Cambridge University Press, First Edition, January 2000.

[8] Microsoft. *Universal Plug and Play Architecture, V1.0*, Jun 2000.

[9] S. Microsystems. *The Jini Architecture Specification, version 2.0*, June 2003.

[10] T. C. Ruys. Low-fat recipes for spin. In *Proceedings of the 7th International SPIN Workshop on SPIN Model Checking and Software Verification*, pages 287–321, London, UK, 2000. Springer-Verlag.

[11] V. Sundramoorthy, J. Scholten, P. G. Jansen, and P. H. Hartel. Service discovery at home. In *4th Int. Conf. on Information, Communications & Signal Processing and 4th IEEE Pacific-Rim Conf. On Multimedia (ICICS/PCM)*, page 1929. IEEE Computer Society Press, December 2003.

[12] V. Sundramoorthy and G. van de Glind. Frodo high-level and detailed design specifications –version 1.0. Technical Report TR-CTIT-06-25, Enschede, June 2006.

Appendix

FRODO Dt-Spin Models

Version 1.0
(subject to change)

June 2006

Authors:
V.Sundramoorthy[1]
Ceryen Tan[2]

[1] University of Twente, the Netherlands
[2] Massachusetts Institute of Technology, USA

Contact: vasughi@cs.utwente.nl

Dt-Spin is a model-checker from Eindhoven University of Technology
http://www.win.tue.nl/~dragan/DTSpin.html
Based on the Spin tool by Bell Labs
www.spinroot.com/

Abstract

We present the FRODO service discovery protocol models that we verify in Dt-Spin. The models are based on the FRODO Specifications in http://eprints.eemcs.utwente.nl/2710/.
FRODO classifies devices into 3C, 3D, and 300D, based on increasing resources. The models are verified against the Service Discovery Principles.

Contents

Exhaustive Verification

Leader election with no message loss 4

Leader election with message loss 13

Backup assignment with no message loss – version 1 14

Backup assignment with no message loss – version 2 34

Backup takeover (Central failure) with no message loss 49

Leader election with periodic announcement 55

Leader election request with message loss 61

Central handover (new node added) with no message loss 72

Registration for 300D with no message loss 79

Registration for 3D with no message loss 85

Registration for 3D with message loss 91

Registration for 3C with no message loss 97

Registration for 3C with message loss 102

3-party Consistency Maintenance with no message loss 108

2-party Consistency Mainetance with no message loss 115

Supertrace

Multiple Leaders Elected (Central Negotiation) with no message loss 123

Multiple Leaders Elected (Central Negotitation) with message loss 142

Backup Assignment with message loss – version 1 161

Backup Assignment with message loss – version 2 183

Registration for 300D with message loss 201

3-party Consistency Maintenance with message loss 212

2-party Consistency Maintenance with message loss 219

Notes:

We call the Registry with the previous name, Central.

Some of the names of messages are not be exactly as in the Rapide specification (but are shortened for ease of modeling.

```
//Leader Election with no message loss
//Ceryen Tan, Vasughi Sundramoorthy
// 8/20 - Clear buffer error
// 8/17 - Message renaming
// 8/16 - Cleanup attempt

#include "dtime.h"

#define N 3
#define MAX_CH 10
#define MAX_LOSS 4

#define Unicast 0
#define Broadcast 1

mtype = {
		MyResource, IAmCentral, CentralNego, LeaderElect,
		NewCentralAssign, CentralAssignAck,
		SrvRegRqst
	} ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
chan nego_list[N] = [N] of {byte, bit, bit } ;

byte nr_leader = 0 ;
byte rightAssigned = 0 ;

byte lossCounters[7] ;

inline clearBuffer()
{
	do
	:: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
	:: empty(from_mail_to_i) -> break
	od
}

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0
}

inline clearVariables()
{
	announceReceived = 0 ;
	negoReceived = 0 ;

	waitMode = 0 ;
	centralNego = 0 ;
```

```
		waitAnnounce = 0 ;

		waitAppointAck = 0 ;
		lastAppointed = 255 ;

		reset( waitDelay ) ;
		reset( nodeDelay ) ;
		reset( centralNegoMax ) ;
		reset( centralAppointDelay ) ;

		do
		:: my_nego_list ? _, _, _
		:: empty( my_nego_list ) -> break
		od
}

inline getType( msgType )
{
		byte type ;

		if
		:: msgType == MyResource -> type = 0
		:: msgType == IAmCentral -> type = 1
		:: msgType == CentralNego -> type = 2
		:: msgType == LeaderElect -> type = 3
		:: msgType == NewCentralAssign -> type = 4
		:: msgType == SrvRegRqst -> type = 5
		:: msgType == CentralAssignAck -> type = 6
		:: else -> type = 255
		fi
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
		getType( msgType ) ;

		atomic
		{
			if
			:: transType == Unicast ->
				if
				:: type == 255 || lossCounters[type] >= MAX_LOSS ->
					from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
				:: else ->
					if
					:: from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
					:: true -> printf( "MSC: UNICAST: %d, %d, %d\n", aMsg, aSrc, dest )
; lossCounters[type]++
					fi
				fi ;
			:: transType == Broadcast ->
				dest = 0 ;

				do
				:: dest < N ->
```

```
                                        if
                                        :: dest != id ->
                                                if
                                                :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                                        from_mail[dest] ! msgType, transType, aMsg, aSrc,
dest
                                                :: else ->
                                                        if
                                                        :: from_mail[dest] ! msgType, transType, aMsg,
aSrc, dest
                                                        :: true -> printf( "MSC: BROADCAST: %d, %d,
%d\n", aMsg, aSrc, dest ) ;
                                                                lossCounters[type]++
                                                        fi
                                                fi ;
                                        :: else
                                        fi ;

                                        dest++
                                :: dest >= N -> dest = 0 ; break
                                od ;
                        fi ;

                        type = 0
        }
}

proctype Node(byte id, rank)
{
        /* Mail */
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* Central nego table */
        chan my_nego_list = nego_list[id] ;
        xr my_nego_list ;
        xs my_nego_list ;

        bit announceReceived ;
        bit negoReceived ;

        /* Messages */
        mtype msgType ;
        bit transType ;
        byte msg ;
        byte src ;
        byte dest ;

        /* Central flags */
        byte waitMode ;
        bit centralNego ;

        /* Normal flags */
        bit waitAnnounce ;

        /* Central appoint */
```

```
        byte waitAppointAck ;
        byte lastAppointed ;

        /* Timers */
        timer waitDelay ;
        timer nodeDelay ;
        timer centralNegoMax ;
        timer centralAppointDelay ;

        printf("MSC: START\n") ;
        atomic { clearBuffer() } ;

start:
        atomic { clearVariables() } ;

        mode0:
                sendMessage( MyResource, Broadcast, rank, id, 255 ) ;
                delay( nodeDelay, 1 ) ;
                nr_leader++ ;
                goto mode1 ;

        mode1:
                set( waitDelay, 30 ) ;

                do
                :: waitDelay.val > 0 ->
                        if
                        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                                delay( nodeDelay, 1 ) ;

                                atomic
                                {
                                        if
                                        :: transType == Broadcast ->
                                                if
                                                :: src != id ->
                                                        if
                                                        :: msgType == MyResource ->
                                                                printf( "MSC: %d\n", msg ) ;
                                                                if
                                                                :: msg > rank ->
                                                                        printf( "MSC: LOST\n" ) ;

                                                                        clearVariables() ;
                                                                        waitAnnounce = 1 ;

                                                                        nr_leader-- ;
                                                                        goto mode3
                                                                :: else
                                                                fi

                                                        :: else
                                                        fi
                                                :: else
                                                fi
```

```
                                        :: transType == Unicast ->
                                                if
                                                :: msgType == NewCentralAssign ->
                                                        printf( "MSC: WON\n" ) ;
                                                        sendMessage( CentralAssignAck, Unicast,
0, id, src ) ;

                                                        goto mode2

                                                :: msgType == SrvRegRqst ->
                                                        printf( "MSC: LOST\n" ) ;

                                                        clearVariables() ;

                                                        nr_leader-- ;
                                                        goto mode3

                                                :: else
                                                fi
                                        fi ;

                                        clearMail()
                                }

                        :: empty(from_mail_to_i) -> delay( nodeDelay, 1 )
                        fi

                :: waitDelay.val <= 0 ->
                        printf( "MSC: WON\n") ;
                        delay( nodeDelay, 1 ) ;
                        goto mode2
                od ;

        mode2:
                printf("MSC: CENTRAL\n");

                if
                :: id == 0 -> rightAssigned = 1
                :: else
                fi ;

end1:           do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast ->
                                        if
                                        :: src != id ->
                                                if
                                                :: msgType == IAmCentral ->
                                                        announceReceived = 0 ;
                                                        negoReceived = 0 ;
```

```
                                                    if
                                                    :: my_nego_list ?? [eval(src),
announceReceived, negoReceived] ->
                                                            my_nego_list ?? src,
announceReceived, negoReceived
                                                    :: else
                                                    fi ;

                                                    if
                                                    :: !(announceReceived == 1 &&
negoReceived == 1) ->
                                                            sendMessage( CentralNego,
Unicast, rank, id, src ) ;
                                                            waitMode = 2 ;
                                                            centralNego = 1 ;
                                                            reset( waitDelay ) ;
                                                            set( centralNegoMax, 30 ) ;
                                                    :: else
                                                    fi ;

                                                    announceReceived = 1 ;
                                                    negoReceived = 0 ;
                                                    my_nego_list ! src, announceReceived,
negoReceived ;

                                                    announceReceived = 0 ;
                                                    negoReceived = 0

                                            :: msgType == LeaderElect ->
                                                    sendMessage( IAmCentral, Broadcast, 0, id,
255 )

                                            :: msgType == MyResource ->
                                                    if
                                                    :: msg > rank ->
                                                            sendMessage( NewCentralAssign,
Unicast, 0, id, src ) ;

                                                            waitAppointAck = 1 ;
                                                            lastAppointed = src ;
                                                            set( centralAppointDelay, 30 )

                                                    :: msg <= rank ->
                                                            sendMessage( SrvRegRqst,
Unicast, 0, id, src )
                                                    fi

                                            :: else
                                            fi
                                    :: else
                                    fi

                            :: transType == Unicast ->
                                    if
                                    :: msgType == CentralAssignAck ->
                                            if
```

```
                                        :: src == lastAppointed ->
                                                clearVariables() ;
                                                waitAnnounce = 1 ;

                                                if
                                                :: id == 0 -> rightAssigned = 0
                                                :: else
                                                fi ;

                                                nr_leader-- ;
                                                goto mode3
                                        :: else
                                        fi

                                :: msgType == CentralNego ->
                                        announceReceived = 0 ;
                                        negoReceived = 0 ;

                                        if
                                        :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->

                                                my_nego_list ?? src, announceReceived,

negoReceived

                                        :: else
                                        fi ;

                                        if
                                        :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->

                                                sendMessage( CentralNego, Unicast, rank,

id, src ) ;

                                        :: else
                                        fi ;

                                        negoReceived = 1 ;
                                        my_nego_list ! src, announceReceived,

negoReceived ;


                                        announceReceived = 0 ;
                                        negoReceived = 0 ;

                                        if
                                        :: msg > rank ->
                                                printf("MSC: LOST\n") ;

                                                clearVariables() ;
                                                waitAnnounce = 1 ;

                                                if
                                                :: id == 0 -> rightAssigned = 0
                                                :: else
                                                fi ;

                                                nr_leader-- ;
                                                goto mode3
```

```
                                        :: else ->
                                                if
                                                :: centralNego == 0 ->
                                                        centralNego = 1 ;
                                                        waitMode = 2 ;
                                                        reset( waitDelay ) ;
                                                        set( centralNegoMax, 30 )
                                                :: else
                                                fi
                                        fi

                                :: else
                                fi
                        fi ;

                        clearMail()
                }

        /*** Announce modes ***/
        :: waitMode == 0 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 1, id, 255 ) ;
                waitMode++ ;
                set( waitDelay, 10 )
        :: waitMode == 1 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 1, id, 255 ) ;
                waitMode++ ;
                set( waitDelay, 10 )
        :: centralNego == 0 && waitMode == 2 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 1, id, 255 ) ;
                set( waitDelay, 30 )

        /*** Central negotiation ***/
        :: centralNego == 1 && centralNegoMax.val <= 0 ->
                if
                :: from_mail_to_i ?? [eval(IAmCentral), transType, msg, src, dest] ||
                        from_mail_to_i ?? [eval(CentralNego), transType, msg, src, dest] ->
                                set( centralNegoMax, 30 )
                :: else ->
                        centralNego = 0 ;
                        waitMode = 2 ;
                        reset( waitDelay ) ;

                        do
                        :: my_nego_list ? _, _, _
                        :: empty( my_nego_list ) -> break
                        od
                fi

        /*** Central appoint ***/
        :: waitAppointAck == 1 && centralAppointDelay.val <= 0 ->
                sendMessage( NewCentralAssign, Unicast, 0, id, lastAppointed ) ;
                waitAppointAck = 2 ;
                set( centralAppointDelay, 30 ) ;
```

```
                :: waitAppointAck == 2 && centralAppointDelay.val <= 0 ->
                        waitAppointAck = 0 ;
                        lastAppointed = 255
                od ;

        mode3:
                printf("MSC: NORMAL\n");
                if
                :: waitAnnounce == 1 -> set( waitDelay, 60 )
                :: else
                fi ;

end2:           do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast ->
                                        if
                                        :: src != id ->
                                                if
                                                :: msgType == IAmCentral ->
                                                        waitAnnounce = 0 ;
                                                        reset( waitDelay )

                                                :: msgType == LeaderElect ->
                                                        waitAnnounce = 2 ;
                                                        set( waitDelay, 30 )

                                                :: else
                                                fi
                                        :: else
                                        fi

                                :: transType == Unicast -> skip
                                fi ;

                                clearMail()
                        }

                /*** Initial wait for announcement ***/
                :: waitAnnounce == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( LeaderElect, Broadcast, 0, id, 255 ) ;
                        waitAnnounce = 2 ;
                        set( waitDelay, 30 )
                :: waitAnnounce == 2 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        goto start
                od
}

init
{
```

```
	atomic
	{
		run Node(0, 10) ;
		run Node(1, 4) ;
		run Node(2, 8)
	}
}


// Leader election with message loss
// Ceryen Tan, Vasughi Sundramoorthy
// 8/20 - Clear buffer error
// 8/17 - Message renaming
// 8/16 - Cleanup attempt

#include "dtime.h"

#define N 3
#define MAX_CH 10
#define MAX_LOSS 4

#define Unicast 0
#define Broadcast 1

mtype = {
		MyResource, IAmCentral, CentralNego, LeaderElect,
		NewCentralAssign, CentralAssignAck,
		SrvRegRqst
	} ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
chan nego_list[N] = [N] of {byte, bit, bit } ;

byte nr_leader = 0 ;
byte rightAssigned = 0 ;

byte lossCounters[7] ;

inline clearBuffer()
{
	do
	:: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
	:: empty(from_mail_to_i) -> break
	od
}

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0
}

inline clearVariables()
{
```

```
        announceReceived = 0 ;
        negoReceived = 0 ;

        waitMode = 0 ;
        centralNego = 0 ;

        waitAnnounce = 0 ;

        waitAppointAck = 0 ;
        lastAppointed = 255 ;

        reset( waitDelay ) ;
        reset( nodeDelay ) ;
        reset( centralNegoMax ) ;
        reset( centralAppointDelay ) ;

        do
        :: my_nego_list ? _, _, _
        :: empty( my_nego_list ) -> break
        od
}

inline getType( msgType )




//Backup Assignment with no message loss
// Ceryen Tan, Vasughi Sundramoorthy
// 8/20 - Clear buffer error
// 8/17 - Message renaming
// 8/16 - Cleanup attempt

#include "dtime.h"

#define N 3
#define MAX_CH 6

#define Unicast 0
#define Broadcast 1

mtype = {
                MyResource, IAmCentral, CentralNego, LeaderElect, MoreCentrals,
                NewCentralAssign, CentralAssignAck,
                BackupAssign, BackupAssignAck, HelloCentral, HelloDevice, BackupCancel,
                MyResRqst, MyResReply, SrvRegRqst,
                SrvSearch, SrvSearchReply
        } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
chan nego_list[N] = [N] of {byte, bit, bit } ;

byte nr_leader = 0 ;
byte nr_backup = 0 ;
byte rightAssigned = 0 ;
byte ranklistSize = 0 ;
```

```
typedef rankEntry
{
	byte id, rank ;
} ;


inline clearBuffer()
{
	atomic
	{
		do
		:: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
		:: empty(from_mail_to_i) -> break
		od
	}
}

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0
}

inline checkRanklist( anID )
{
	atomic
	{
		findInRanklist( anID ) ;

		if
		:: tempLoc == 255 -> sendMessage( MyResRqst, Unicast, 0, id, anID )
		:: else -> tempLoc = 0
		fi
	}
}

inline clearRanklist()
{
	atomic
	{
		counter = 0 ;
		do
		:: counter < N -> rankList[counter].id = 255 ; rankList[counter].rank = 255 ; counter++
		:: counter == N -> counter = 0; break
		od ;

		if
		:: id == 0 -> ranklistSize = 0
		:: else
		fi
	}
}
```

```
inline clearVariables()
{
	announceReceived = 0 ;
	negoReceived = 0 ;

	mode = 0 ;
	waitMode = 0 ;
	centralNego = 0 ;

	waitAnnounce = 0 ;
	sendSrvSearch = 0 ;
	waitSrvSearchReply = 0 ;

	waitAppointAck = 0 ;
	lastAppointed = 255 ;

	lastCentral = 255 ;
	lastBackup = 255 ;
	sendHelloBackup = 0 ;
	sendHelloCentral = 0 ;

	assignBackup = 0 ;
	backupAssigned = 0 ;
	waitBackupReply = 0 ;
	attemptedAssign = 0 ;

	backupRank = 0 ;
	ranklistEmpty = 0 ;

	reset( waitDelay ) ;
	reset( nodeDelay ) ;
	reset( backupDelay ) ;
	reset( centralNegoMax ) ;
	reset( centralAppointDelay ) ;

	do
	:: nempty( my_nego_list ) -> my_nego_list ? _, _, _
	:: empty( my_nego_list ) -> break
	od
}

inline findHighest()
{
	atomic
	{
		counter = 0 ;
		aRank = 0 ;
		anID = 255 ;

		do
		:: counter < N ->
			if
			:: rankList[counter].rank != 255 && rankList[counter].rank > aRank ->
				aRank = rankList[counter].rank ;
				anID = rankList[counter].id
```

```
                        :: else
                        fi ;
                        counter++
                :: counter == N -> counter = 0 ; break
                od
        }
}

inline findInRanklist( anID )
{
        atomic
        {
                tempLoc = 255 ;
                counter = 0 ;
                do
                :: counter < N ->
                        if
                        :: rankList[counter].id == anID -> tempLoc = counter ; break
                        :: else
                        fi ;
                        counter++
                :: counter == N -> counter = 0 ; break
                od
        }
}

inline removeEntry( anID )
{
        atomic
        {
                counter = 0 ;

                do
                :: counter < N ->
                        if
                        :: rankList[counter].id == anID ->
                                rankList[counter].id = 255;
                                rankList[counter].rank = 255;

                                if
                                :: id == 0 -> ranklistSize--
                                :: else
                                fi
                        :: else
                        fi ;
                        counter++
                :: counter == N -> counter = 0 ; break
                od
        }
}

inline updateRanklist( aRank, anID )
{
        atomic
        {
                counter = 0 ;
```

```
		tempLoc = 255 ;

		do
		:: counter < N ->
			if
			:: rankList[counter].id == anID -> tempLoc = counter ; break
			:: else ->
				if
				:: tempLoc == 255 &&
					rankList[counter].rank == 255 &&
					rankList[counter].id == 255 ->
					tempLoc = counter
				:: else -> counter++
				fi
			fi ;
		:: counter == N -> break
		od ;

		if
		:: tempLoc == 255 -> printf("MSC: Full Ranklist\n")
		:: else ->
			if
			:: id == 0 && rankList[tempLoc].id != anID -> ranklistSize++
			:: else
			fi ;

			rankList[tempLoc].rank = aRank ;
			rankList[tempLoc].id = anID
		fi ;

		counter = 0 ;
		tempLoc = 0 ;
	}
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
	atomic
	{
		if
		:: transType == Unicast -> from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
		:: transType == Broadcast ->
			dest = 0 ;

			do
			:: dest < N ->
				if
				:: dest != id -> from_mail[dest] ! msgType, transType, aMsg, aSrc, dest
;
				:: else
				fi ;

				dest++
			:: dest >= N -> dest = 0 ; break
			od
		fi
```

```
		}
}

proctype Node(byte id, rank)
{
		/* Mail */
		chan from_mail_to_i = from_mail[id] ;
		xr from_mail_to_i ;


		/* Messages */
		mtype msgType ;
		bit transType ;
		byte msg ;
		byte src ;
		byte dest ;

		/* Central negotiation table */
		chan my_nego_list = nego_list[id] ;
		xr my_nego_list ;
		xs my_nego_list ;

		bit announceReceived ;
		bit negoReceived ;

		/* Ranklist */
		rankEntry rankList[N] ;
		byte counter ;
		byte tempLoc ;
		byte aRank, anID ;

		/* Central flags */
		byte mode ;
		byte waitMode ;
		bit centralNego ;

		/* Normal flags */
		byte waitAnnounce ;
		byte sendSrvSearch ;
		byte waitSrvSearchReply ;

		/* Central appoint */
		byte waitAppointAck ;
		byte lastAppointed ;

		/* Backup */
		byte lastCentral ;
		byte lastBackup ;
		byte sendHelloBackup ;
		byte sendHelloCentral ;

		byte assignBackup ;
		byte backupAssigned ;
		byte waitBackupReply ;
		bit attemptedAssign ;
```

```
	byte backupRank ;
	byte ranklistEmpty ;

	/* Timers */
	timer waitDelay ;
	timer nodeDelay ;
	timer backupDelay ;
	timer centralNegoMax ;
	timer centralAppointDelay ;

	printf("MSC: START\n") ;
	atomic { clearBuffer() } ;

start:
	atomic { clearRanklist() } ;

	do
	:: mode == 0 ->                                 /***** Startup *****/
		sendMessage( MyResource, Broadcast, rank, id, 255 ) ;
		delay( nodeDelay, 1 ) ;
		nr_leader++ ;
		mode++

	:: mode == 1 ->                                 /***** Listen mode *****/
		set( waitDelay, 30 ) ;

		do
		:: waitDelay.val > 0 ->
			if
			:: from_mail_to_i ? msgType, transType, msg, src, dest ->
				delay( nodeDelay, 1 ) ;

				atomic
				{
					if
					:: transType == Broadcast ->
						if
						:: src != id ->
							if
							:: msgType == MyResource ->
								printf( "MSC: %d\n", msg ) ;
								if
								:: msg > rank ->
									printf( "MSC: LOST\n" ) ;

									clearRanklist() ;
									clearVariables() ;
									waitAnnounce = 1 ;

									mode = 3 ;
									nr_leader-- ;
									break
								:: else -> updateRanklist( msg, src )
;
								fi
```

```
                                        :: else
                                        fi
                                :: else
                                fi

                        :: transType == Unicast ->
                                if
                                :: msgType == BackupAssign ->
                                        printf( "MSC: LOST\n" ) ;
                                        sendMessage( BackupAssignAck, Unicast,
0, id, src ) ;

                                        clearRanklist() ;
                                        clearVariables() ;
                                        lastCentral = src ;

                                        mode = 4 ;
                                        nr_leader-- ;
                                        nr_backup++ ;
                                        break

                                :: msgType == NewCentralAssign ->
                                        printf( "MSC: WON\n" ) ;
                                        sendMessage( CentralAssignAck, Unicast,
0, id, src ) ;

                                        clearVariables() ;
                                        lastBackup = src ;
                                        backupRank = msg ;
                                        sendHelloBackup = 1 ;
                                        assignBackup = 0 ;
                                        backupAssigned = 1 ;

                                        mode = 2 ;
                                        break

                                :: msgType == SrvRegRqst ->
                                        printf( "MSC: LOST\n" ) ;

                                        clearRanklist() ;
                                        clearVariables() ;

                                        mode = 3 ;
                                        nr_leader-- ;
                                        break

                                :: else
                                fi
                        fi ;

                        clearMail()
                }

        :: empty(from_mail_to_i) -> delay( nodeDelay, 1 )
        fi
```

```
                :: waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        printf( "MSC: WON\n") ;

                        mode = 2 ;
                        break
                od

        :: mode == 2 ->                         /***** Central mode *****/
                printf("MSC: CENTRAL\n");

                if
                :: id == 0 -> rightAssigned = 1
                :: else
                fi ;

end1:           do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast ->
                                        if
                                        :: src != id ->
                                                if
                                                :: msgType == IAmCentral ->
                                                        announceReceived = 0 ;
                                                        negoReceived = 0 ;

                                                        if
                                                        :: my_nego_list ?? [eval(src),
announceReceived, negoReceived] ->
                                                                my_nego_list ?? src,
announceReceived, negoReceived
                                                        :: else
                                                        fi ;

                                                        if
                                                        :: !(announceReceived == 1 &&
negoReceived == 1) ->
                                                                sendMessage( CentralNego,
Unicast, rank, id, src ) ;

                                                                centralNego = 1 ;
                                                                waitMode = 2 ;
                                                                reset( waitDelay ) ;
                                                                set( centralNegoMax, 30 ) ;
                                                        :: else
                                                        fi ;

                                                        announceReceived = 1 ;
                                                        negoReceived = 0 ;
                                                        my_nego_list ! src, announceReceived,
negoReceived ;
```

```
                                        announceReceived = 0 ;
                                        negoReceived = 0

                                    :: msgType == LeaderElect ->
                                        sendMessage( IAmCentral, Broadcast, 0, id,
255 ) ;
                                        checkRanklist( src )

                                    :: msgType == MyResource ->
                                        if
                                        :: msg > rank ->
                                            sendMessage( NewCentralAssign,
Unicast, 0, id, src ) ;

                                            waitAppointAck = 1 ;
                                            lastAppointed = src ;
                                            set( centralAppointDelay, 30 )

                                        :: msg <= rank ->
                                            sendMessage( SrvRegRqst,
Unicast, 0, id, src )
                                        fi ;

                                        if
                                        :: msg > backupRank && (backupAssigned
== 1 || ranklistEmpty == 1) ->
                                            if
                                            :: lastBackup != 255 ->
                                                sendMessage(
BackupCancel, Unicast, 0, id, lastBackup ) ;
                                            :: else
                                            fi ;

                                            sendMessage( BackupAssign,
Unicast, 0, id, src ) ;
                                            lastBackup = src ;
                                            backupRank = msg ;
                                            waitBackupReply = 1 ;
                                            backupAssigned = 0 ;
                                            sendHelloBackup = 0 ;
                                            assignBackup = 0 ;
                                            attemptedAssign = 1 ;
                                            set( backupDelay, 15 )
                                        :: else
                                        fi ;

                                        ranklistEmpty = 0 ;
                                        updateRanklist( msg, src )

                                    :: else -> checkRanklist( src )
                                    fi
                            :: else
                            fi

                        :: transType == Unicast ->
```

```
                                        if
                                        :: msgType == BackupAssignAck ->
                                                if
                                                :: src == lastBackup ->
                                                        backupAssigned = 1 ;
                                                        waitBackupReply = 0 ;
                                                        sendHelloBackup = 1 ;
                                                        assignBackup = 0 ;
                                                        set( backupDelay, 15 )
                                                :: else ->
                                                        sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
                                                        checkRanklist( src )
                                                fi

                                        :: msgType == CentralAssignAck ->
                                                if
                                                :: src == lastAppointed ->
                                                        if
                                                        :: backupAssigned == 1 ->
                                                                sendMessage( BackupCancel,
Unicast, 0, id, lastBackup)
                                                        :: else
                                                        fi ;

                                                        clearVariables() ;
                                                        lastCentral = src ;

                                                        if
                                                        :: id == 0 -> rightAssigned = 0
                                                        :: else
                                                        fi ;

                                                        mode = 4 ;
                                                        nr_leader-- ;
                                                        break
                                                :: else
                                                fi

                                        :: msgType == CentralNego ->
                                                announceReceived = 0 ;
                                                negoReceived = 0 ;

                                                if
                                                :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                                                        my_nego_list ?? src, announceReceived,
negoReceived
                                                :: else
                                                fi ;

                                                if
                                                :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                                        sendMessage( CentralNego, Unicast, rank,
id, src ) ;
```

```
                                        :: else
                                        fi ;

                                        negoReceived = 1 ;
                                        my_nego_list ! src, announceReceived,
negoReceived ;


                                        announceReceived = 0 ;
                                        negoReceived = 0 ;

                                        if
                                        :: msg > rank ->
                                                printf("MSC: LOST\n") ;
                                                if
                                                :: backupAssigned == 1 ->
                                                        sendMessage( BackupCancel,
Unicast, 0, id, lastBackup ) ;
                                                :: else
                                                fi ;

                                                clearRanklist() ;
                                                clearVariables() ;
                                                waitAnnounce = 1 ;

                                                if
                                                :: id == 0 -> rightAssigned = 0
                                                :: else
                                                fi ;

                                                mode = 3 ;
                                                nr_leader-- ;
                                                break
                                        :: else ->
                                                if
                                                :: centralNego == 0 ->
                                                        centralNego = 1 ;
                                                        waitMode = 2 ;
                                                        reset( waitDelay ) ;
                                                        set( centralNegoMax, 30 )
                                                :: else
                                                fi ;

                                                updateRanklist( msg, src )
                                        fi ;

                                        if
                                        :: msg > backupRank && (backupAssigned == 1 ||
ranklistEmpty == 1) ->
                                                if
                                                :: backupAssigned == 255 ->
                                                        sendMessage( BackupCancel,
Unicast, 0, id, lastBackup ) ;
                                                :: else
                                                fi ;
```

```
                                        sendMessage( BackupAssign, Unicast, 0, id,
src ) ;
                                        lastBackup = src ;
                                        backupRank = msg ;
                                        waitBackupReply = 1 ;
                                        backupAssigned = 0 ;
                                        sendHelloBackup = 0 ;
                                        assignBackup = 0 ;
                                        attemptedAssign = 1 ;
                                        set( backupDelay, 15 )
                                :: else
                                fi ;

                                ranklistEmpty = 0

                        :: msgType == HelloCentral ->
                                if
                                :: src == lastBackup ->
                                        sendHelloBackup = 1 ;
                                        set( backupDelay, 15 )
                                :: else ->
                                        sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
                                        checkRanklist( src )
                                fi

                        :: msgType == MoreCentrals ->
                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                                        my_nego_list ?? src, announceReceived,
negoReceived
                                :: else
                                fi ;

                                if
                                :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                        sendMessage( CentralNego, Unicast, rank,
id, msg ) ;
                                :: else
                                fi ;

                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: src != lastBackup ->
                                        sendMessage( BackupCancel, Unicast, 0, id,
src )
                                :: else ->
                                fi ;
```

```
                                        checkRanklist( src )

                                :: msgType == MyResReply ->
                                        if
                                        :: msg > rank ->
                                                sendMessage( NewCentralAssign, Unicast,
rank, id, src ) ;

                                                waitAppointAck = 1 ;
                                                lastAppointed = src ;
                                                set( centralAppointDelay, 30 )

                                        :: else

                                        fi ;

                                        if
                                        :: msg > backupRank && (backupAssigned == 1 ||
ranklistEmpty == 1) ->
                                                if
                                                :: backupAssigned == 1 ->
                                                        sendMessage( BackupCancel,
Unicast, 0, id, lastBackup ) ;
                                                :: else
                                                fi ;

                                                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;
                                                lastBackup = src ;
                                                backupRank = msg ;
                                                waitBackupReply = 1 ;
                                                sendHelloBackup = 0 ;
                                                assignBackup = 0 ;
                                                attemptedAssign = 0 ;
                                                set( backupDelay, 15 )
                                        :: else
                                        fi ;

                                        ranklistEmpty = 0 ;
                                        updateRanklist( msg, src )

                                :: msgType == SrvSearch ->
                                        sendMessage( SrvSearchReply, Unicast, 0, id, src ) ;
                                        checkRanklist( src )

                                :: else -> checkRanklist( src )
                                fi
                        fi ;

                        clearMail()
                }

        /*** Announce modes ***/
        :: waitMode == 0 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;
```

```
		waitMode++ ;
		set( waitDelay, 30 )
	:: waitMode == 1 && waitDelay.val <= 0 ->
		delay( nodeDelay, 1 ) ;
		sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;
		waitMode++ ;

		if
		:: attemptedAssign == 0 -> assignBackup = 1 ; attemptedAssign = 1
		:: else
		fi ;

		set( waitDelay, 30 )
	:: centralNego == 0 && waitMode == 2 && waitDelay.val <= 0 ->
		delay( nodeDelay, 1 ) ;
		sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;

		if
		:: attemptedAssign == 0 -> assignBackup = 1 ; attemptedAssign = 1
		:: else
		fi ;

		set( waitDelay, 30 )

	/*** Central Negotiation ***/
	:: centralNego == 1 && centralNegoMax.val <= 0 ->
		if
		:: from_mail_to_i ?? [eval(IAmCentral), transType, msg, src, dest] ||
			from_mail_to_i ?? [eval(CentralNego), transType, msg, src, dest] ->
				set( centralNegoMax, 30 )
		:: else ->
			centralNego = 0 ;		/* Do an announce with 0 first */
			waitMode = 2 ;
			reset( waitDelay ) ;

			do
			:: nempty( my_nego_list ) -> my_nego_list ? _, _, _
			:: empty( my_nego_list ) -> break
			od
		fi

	/*** Backup assignment ***/
	:: centralNego == 0 && assignBackup == 1 ->
		delay( nodeDelay, 1 ) ;

		atomic
		{
			findHighest() ;

			if
			:: anID == 255 ->
				printf("MSC: Empty List\n") ;
				assignBackup = 0 ;
				lastBackup = 255 ;
				backupRank = 0 ;
				ranklistEmpty = 1 ;
```

```
                        sendHelloBackup = 0 ;
                        waitBackupReply = 0 ;
                        backupAssigned = 0 ;
                :: else ->
                        printf("MSC: Backup %d\n", anID) ;
                        sendMessage( BackupAssign, Unicast, 0, id, anID ) ;
                        lastBackup = anID ;
                        backupRank = aRank ;
                        assignBackup = 0 ;
                        waitBackupReply = 1 ;
                        sendHelloBackup = 0 ;
                        backupAssigned = 0 ;
                        set( backupDelay, 15 )
                fi
        }
:: waitBackupReply == 1 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( BackupAssign, Unicast, 0, id, lastBackup ) ;
        waitBackupReply = 2 ;
        assignBackup = 0 ;
        backupAssigned = 0 ;
        sendHelloBackup = 0 ;
        set( backupDelay, 15 )
:: waitBackupReply == 2 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        removeEntry( lastBackup ) ;
        waitBackupReply = 0 ;
        lastBackup = 255 ;
        backupRank = 0 ;
        assignBackup = 1 ;
        backupAssigned = 0 ;
        sendHelloBackup = 0

/*** Backup Polling ***/
:: sendHelloBackup == 1 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
        sendHelloBackup = 2 ;
        set( backupDelay, 15 )
:: sendHelloBackup == 2 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
        sendHelloBackup = 3 ;
        set( backupDelay, 15 )
:: sendHelloBackup == 3 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        removeEntry( lastBackup ) ;
        lastBackup = 255 ;
        backupRank = 0 ;
        assignBackup = 1 ;
        backupAssigned = 0 ;
        sendHelloBackup = 0 ;
        waitBackupReply = 0

/*** Central Appointment ***/
:: waitAppointAck == 1 && centralAppointDelay.val <= 0 ->
```

```
                        sendMessage( NewCentralAssign, Unicast, rank, id, lastAppointed ) ;
                        waitAppointAck = 2 ;
                        set( centralAppointDelay, 30 ) ;
                :: waitAppointAck == 2 && centralAppointDelay.val <= 0 ->
                        waitAppointAck = 0 ;
                        lastAppointed = 255
                od

        :: mode == 3 ->                         /***** Normal mode *****/
                printf("MSC: NORMAL\n");
                if
                :: waitAnnounce == 1 -> set( waitDelay, 60 )
                :: else
                fi ;

end2:           do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast ->
                                        if
                                        :: src != id ->
                                                if
                                                :: msgType == IAmCentral ->
                                                        waitAnnounce = 0 ;
                                                        lastCentral = src ;

                                                :: msgType == LeaderElect ->
                                                        waitAnnounce = 2 ;
                                                        set( waitDelay, 30 )

                                                :: else
                                                fi
                                        :: else
                                        fi

                                :: transType == Unicast ->
                                        if
                                        :: msgType == BackupAssign ->
                                                if
                                                :: lastCentral != 255 && src != lastCentral ->
                                                        sendMessage( MoreCentrals, Unicast, src,
id, lastCentral ) ;
                                                        sendMessage( MoreCentrals, Unicast,
lastCentral, id, src )
                                                :: else ->
                                                        sendMessage( BackupAssignAck, Unicast,
0, id, src ) ;

                                                        clearVariables() ;
                                                        lastCentral = src ;

                                                        mode = 4 ;
```

```
                                        nr_backup++ ;
                                        break
                                fi

                        :: msgType == NewCentralAssign ->
                                sendMessage( CentralAssignAck, Unicast, 0, id, src )
;

                                clearVariables() ;
                                lastBackup = src ;
                                backupRank = msg ;
                                sendHelloBackup = 1 ;
                                assignBackup = 0 ;
                                backupAssigned = 1 ;

                                mode = 2 ;
                                nr_leader++ ;
                                break

                        :: msgType == MyResRqst ->
                                sendMessage( MyResReply, Unicast, rank, id, src )

                        :: msgType == SrvSearchReply ->
                                waitSrvSearchReply = 0 ;
                                sendSrvSearch = 1 ;
                                set( waitDelay, 60 )

                        :: else
                        fi
                fi ;

                clearMail()
        }

    /*** Initial wait for announcement ***/
    :: waitAnnounce == 1 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( LeaderElect, Broadcast, 0, id, 255 ) ;
        waitAnnounce = 2 ;
        set( waitDelay, 30 )
    :: waitAnnounce == 2 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        goto start

    /*** Service Searching ***/
    :: waitAnnounce == 0 && sendSrvSearch == 0 && waitSrvSearchReply == 0 ->
        delay( nodeDelay, 1 ) ;
        set( waitDelay, 60 ) ;
        sendSrvSearch = 1
    :: sendSrvSearch == 1 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( SrvSearch, Unicast, 0, id, lastCentral ) ;
        waitSrvSearchReply = 1 ;
        sendSrvSearch = 0 ;
        set( waitDelay, 60 )
    :: waitSrvSearchReply == 1 && waitDelay.val <= 0 ->
```

```
                        delay( nodeDelay, 1 ) ;
                        sendMessage( LeaderElect, Broadcast, 0, id, 255 ) ;
                        waitAnnounce = 2 ;
                        waitSrvSearchReply = 0 ;
                        set( waitDelay, 30 )
                od
        :: mode == 4 ->
                printf("MSC: BACKUP\n") ;
                set( waitDelay, 30 ) ;

end3:           do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast -> skip
                                :: transType == Unicast ->
                                        if
//                                      :: msgType == IAmCentral ->
//                                              if
//                                              :: src != lastCentral ->
//                                                      sendMessage( MoreCentrals, Unicast, src,
id, lastCentral ) ;
//                                                      sendMessage( MoreCentrals, Unicast,
lastCentral, id, src )
//                                              :: else
//                                              fi

                                        :: msgType == BackupAssign ->
                                                if
                                                :: src == lastCentral ->
                                                        sendMessage( BackupAssignAck, Unicast,
0, id, lastCentral ) ;
                                                :: else ->
                                                        sendMessage( MoreCentrals, Unicast, src,
id, lastCentral ) ;
                                                        sendMessage( MoreCentrals, Unicast,
lastCentral, id, src )
                                                fi

                                        :: msgType == BackupCancel ->
                                                clearRanklist() ;
                                                clearVariables() ;

                                                mode = 3 ;
                                                nr_backup-- ;
                                                break

                                        :: msgType == HelloDevice ->
                                                if
                                                :: src == lastCentral ->
                                                        sendMessage( HelloCentral, Unicast, 0, id,
src ) ;
                                                        sendHelloCentral = 1 ;
```

```
                                                        set( waitDelay, 30 )
                                                :: else //->
                                                        sendMessage( MoreCentrals, Unicast, src,
id, lastCentral ) ;
                                                        sendMessage( MoreCentrals, Unicast,
lastCentral, id, src )
                                                fi

                                        :: else
                                        fi
                                fi ;

                                clearMail()
                        }

                /*** Central Polling ***/
                :: sendHelloCentral == 0 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( HelloCentral, Unicast, 0, id, lastCentral ) ;
                        sendHelloCentral = 2 ;
                        set( waitDelay, 30 )
                :: sendHelloCentral == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( HelloCentral, Unicast, 0, id, lastCentral ) ;
                        sendHelloCentral = 2 ;
                        set( waitDelay, 30 )
                :: sendHelloCentral == 2 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;

                        clearVariables() ;

                        mode = 2 ;
                        nr_leader++ ;
                        nr_backup -- ;
                        break
                od
        od
}

init
{
        atomic
        {
                run Node(0, 10) ;
                run Node(1, 4) ;
                run Node(2, 8)
        }
}
```

```
//Backup Assignment with no message loss – second version
// Ceryen Tan, Vasughi Sundramoorthy
// 8/20 - Clear buffer error
// 8/17 - Message renaming
// 8/16 - Cleanup attempt

#include "dtime.h"

#define N 3
#define MAX_CH 6

#define Unicast 0
#define Broadcast 1

mtype = {
		IAmCentral, CentralNego, MoreCentrals,
		NewCentralAssign, CentralAssignAck,
		BackupAssign, BackupAssignAck, HelloCentral, HelloDevice, BackupCancel,
		MyResRqst, MyResReply,
		SrvSearch, SrvSearchReply
	} ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
chan nego_list[N] = [N] of {byte, bit, bit } ;

byte nr_leader = 0 ;
byte nr_backup = 0 ;
byte rightAssigned = 0 ;
byte ranklistSize = 0 ;

typedef rankEntry
{
	byte id, rank ;
} ;

inline checkRanklist( anID )
{
	atomic
	{
		findInRanklist( anID ) ;

		if
		:: tempLoc == 255 -> sendMessage( MyResRqst, Unicast, 0, id, anID )
		:: else -> tempLoc = 0
		fi
	}
}

inline clearBuffer()
{
	atomic
	{
		do
```

```
                :: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
                :: empty(from_mail_to_i) -> break
                od
        }
}

inline clearMail()
{
        msgType = 0 ;
        transType = 0 ;
        msg = 0 ;
        src = 0 ;
        dest = 0
}

inline clearRanklist()
{
        atomic
        {
                counter = 0 ;
                do
                :: counter < N -> rankList[counter].id = 255 ; rankList[counter].rank = 255 ; counter++
                :: counter == N -> counter = 0 ; break
                od ;

                if
                :: id == 0 -> ranklistSize = 0
                :: else
                fi
        }
}

inline clearVariables()
{
        waitAppointAck = 0 ;
        lastAppointed = 255 ;

        lastCentral = 255 ;
        lastBackup = 255 ;
        sendHelloBackup = 0 ;
        sendHelloCentral = 0 ;

        assignBackup = 0 ;
        backupAssigned = 0 ;
        waitBackupReply = 0 ;

        backupRank = 0 ;
        ranklistEmpty = 0 ;

        centralNego = 0 ;

        reset( waitDelay ) ;
        reset( nodeDelay ) ;
        reset( backupDelay ) ;
        reset( centralNegoMax ) ;
        reset( centralAppointDelay ) ;
```

```
	do
	:: nempty( my_nego_list ) -> my_nego_list ? _, _, _
	:: empty( my_nego_list ) -> break
	od
}

inline findHighest()
{
	atomic
	{
		counter = 0 ;
		aRank = 0 ;
		anID = 255 ;

		do
		:: counter < N ->
			if
			:: rankList[counter].rank != 255 && rankList[counter].rank > aRank ->
				aRank = rankList[counter].rank ;
				anID = rankList[counter].id
			:: else
			fi ;
			counter++
		:: counter == N -> counter = 0 ; break
		od
	}
}

inline findInRanklist( anID )
{
	atomic
	{
		tempLoc = 255 ;
		counter = 0 ;
		do
		:: counter < N ->
			if
			:: rankList[counter].id == anID -> tempLoc = counter ; break
			:: else
			fi ;
			counter++
		:: counter == N -> counter = 0 ; break
		od
	}
}

inline removeEntry( anID )
{
	atomic
	{
		counter = 0 ;

		do
		:: counter < N ->
			if
```

```
                        :: rankList[counter].id == anID ->
                                rankList[counter].id = 255 ;
                                rankList[counter].rank = 255 ;

                                if
                                :: id == 0 -> ranklistSize--
                                :: else
                                fi
                        :: else
                        fi ;
                        counter++
                :: counter == N -> counter = 0 ; break
                od
        }
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
        atomic
        {
                if
                :: transType == Unicast -> from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                :: transType == Broadcast ->
                        dest = 0 ;

                        do
                        :: dest < N ->
                                if
                                :: dest != id -> from_mail[dest] ! msgType, transType, aMsg, aSrc, dest
;
                                :: else
                                fi ;

                                dest++
                        :: dest >= N -> dest = 0 ; break
                        od ;
                fi ;
        }
}

inline updateRanklist( aRank, anID )
{
        atomic
        {
                counter = 0 ;
                tempLoc = 255 ;

                do
                :: counter < N ->
                        if
                        :: rankList[counter].id == anID -> tempLoc = counter ; break
                        :: else ->
                                if
                                :: tempLoc == 255 &&
                                        rankList[counter].rank == 255 &&
                                        rankList[counter].id == 255 ->
```

```
                                        tempLoc = counter
                                :: else -> counter++
                                fi
                        fi ;
                :: counter == N -> break
                od ;

                if
                :: tempLoc == 255 -> printf("MSC: Full Ranklist\n")
                :: else ->
                        if
                        :: id == 0 && rankList[tempLoc].id != anID -> ranklistSize++
                        :: else
                        fi ;

                        rankList[tempLoc].rank = aRank ;
                        rankList[tempLoc].id = anID
                fi ;

                counter = 0 ;
                tempLoc = 0
        }
}

proctype Node( byte id, rank, mode )
{
        /* Mail */
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* Messages */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        /* Central negotiation table */
        chan my_nego_list = nego_list[id] ;
        xr my_nego_list ;
        xs my_nego_list ;

        bit announceReceived ;
        bit negoReceived ;

        /* Ranklist */
        rankEntry rankList[N] ;
        byte counter ;
        byte tempLoc ;
        byte aRank, anID ;

        /* Central appoint */
        byte waitAppointAck ;
        byte lastAppointed ;

        /* Backup */
```

```
	byte lastCentral ;
	byte lastBackup ;
	byte sendHelloBackup ;
	bit sendHelloCentral ;

	bit assignBackup ;
	bit backupAssigned ;
	byte waitBackupReply ;

	byte backupRank ;
	bit ranklistEmpty ;

	/* Central negotation */
	bit centralNego ;

	/* Timers */
	timer waitDelay ;
	timer nodeDelay ;
	timer backupDelay ;
	timer centralNegoMax ;
	timer centralAppointDelay ;

	printf("MSC: START\n") ;
	atomic { clearBuffer() } ;

start:
	atomic
	{
		clearRanklist() ;
		clearVariables()
	} ;

	do
	:: mode == 2 ->
		printf("MSC: LEADER\n") ;

		if
		:: id == 0 -> rightAssigned = 1
		:: else
		fi ;

		assignBackup = 1 ;
		set( waitDelay, 30 ) ;

		nr_leader++ ;

		do
		:: from_mail_to_i ? msgType, transType, msg, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: transType == Broadcast ->
					if
					:: src != id ->
```

```
                                        if
                                        :: msgType == IAmCentral ->
                                                announceReceived = 0 ;
                                                negoReceived = 0 ;

                                                if
                                                :: my_nego_list ?? [eval(src),
announceReceived, negoReceived] ->
                                                        my_nego_list ?? src,
announceReceived, negoReceived
                                                :: else
                                                fi ;

                                                if
                                                :: !(announceReceived == 1 &&
negoReceived == 1) ->
                                                        sendMessage( CentralNego,
Unicast, rank, id, src ) ;
                                                        centralNego = 1 ;
                                                        reset( waitDelay ) ;
                                                        set( centralNegoMax, 30 ) ;
                                                :: else
                                                fi ;

                                                announceReceived = 1 ;
                                                negoReceived = 0 ;
                                                my_nego_list ! src, announceReceived,
negoReceived ;

                                                announceReceived = 0 ;
                                                negoReceived = 0

                                        :: else -> checkRanklist( src )
                                        fi
                                :: else
                                fi

                        :: transType == Unicast ->
                                if
                                :: msgType == BackupAssignAck ->
                                        if
                                        :: src == lastBackup ->
                                                assignBackup = 0 ;
                                                backupAssigned = 1 ;
                                                waitBackupReply = 0 ;
                                                sendHelloBackup = 1 ;
                                                set( backupDelay, 15 )
                                        :: else ->
                                                sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
                                                checkRanklist( src )
                                        fi

                                :: msgType == CentralAssignAck ->
                                        if
                                        :: src == lastAppointed ->
```

```
                                        if
                                        :: backupAssigned == 1 ->
                                                sendMessage( BackupCancel,
Unicast, 0, id, lastBackup ) ;
                                        :: else
                                        fi ;

                                        clearVariables() ;
                                        lastCentral = src ;

                                        if
                                        :: id == 0 -> rightAssigned = 0
                                        :: else
                                        fi ;

                                        mode = 4 ;
                                        nr_leader-- ;
                                        break
                                :: else
                                fi

                        :: msgType == CentralNego ->
                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                                        my_nego_list ?? src, announceReceived,
negoReceived
                                :: else
                                fi ;

                                if
                                :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                        sendMessage( CentralNego, Unicast, rank,
id, src ) ;
                                :: else
                                fi ;

                                negoReceived = 1 ;
                                my_nego_list ! src, announceReceived,
negoReceived ;

                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: msg > rank ->
                                        printf("MSC: LOST\n") ;
                                        if
                                        :: backupAssigned == 1 ->
                                                sendMessage( BackupCancel,
Unicast, 0, id, lastBackup )
                                        :: else
```

```
                fi ;

                clearRanklist() ;
                clearVariables() ;

                if
                :: id == 0 -> rightAssigned = 0
                :: else
                fi ;

                mode = 3 ;
                nr_leader-- ;
                break
        :: else ->
                if
                :: centralNego == 0 ->
                        centralNego = 1 ;
                        reset( waitDelay ) ;
                        set( centralNegoMax, 30 )
                :: else
                fi
        fi ;

        if
        :: backupAssigned == 1 && msg > backupRank ->
                sendMessage( BackupCancel, Unicast, 0, id,
lastBackup ) ;
                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;
                lastBackup = src ;
                backupRank = msg ;

                assignBackup = 0 ;
                backupAssigned = 0 ;
                waitBackupReply = 1 ;
                sendHelloBackup = 0 ;
                set( backupDelay, 15 )
        :: else
        fi ;

        if
        :: ranklistEmpty == 1 ->
                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;

                lastBackup = src ;
                backupRank = msg ;
                assignBackup = 0 ;
                waitBackupReply = 1 ;
                backupAssigned = 0 ;
                sendHelloBackup = 0 ;
                set( backupDelay, 15 ) ;
                ranklistEmpty = 0
        :: else
        fi ;
```

```
                                        updateRanklist( msg, src )

                                    :: msgType == HelloCentral ->
                                        if
                                        :: src == lastBackup ->
                                            sendHelloBackup = 1 ;
                                            set( backupDelay, 15 )
                                        :: else ->
                                            sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
                                            checkRanklist( src )
                                        fi

                                    :: msgType == MoreCentrals ->
                                        announceReceived = 0 ;
                                        negoReceived = 0 ;

                                        if
                                        :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                                            my_nego_list ?? src, announceReceived,
negoReceived
                                        :: else
                                        fi ;

                                        if
                                        :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                            sendMessage( CentralNego, Unicast, rank,
id, msg ) ;
                                        :: else
                                        fi ;

                                        announceReceived = 0 ;
                                        negoReceived = 0 ;

                                        if
                                        :: src != lastBackup ->
                                            sendMessage( BackupCancel, Unicast, 0, id,
src )
                                        :: else
                                        fi ;

                                        checkRanklist( src )

                                    :: msgType == MyResReply ->
                                        if
                                        :: msg > rank ->
                                            sendMessage( NewCentralAssign, Unicast,
rank, id, src ) ;

                                            waitAppointAck = 1 ;
                                            lastAppointed = src ;
                                            set( centralAppointDelay, 30 )

                                        :: else
```

```
                                                fi ;

                                                if
                                                :: backupAssigned == 1 && msg > backupRank ->
                                                        sendMessage( BackupCancel, Unicast, 0, id,
lastBackup ) ;
                                                        sendMessage( BackupAssign, Unicast, 0, id,
src ) ;
                                                        lastBackup = src ;
                                                        backupRank = msg ;

                                                        assignBackup = 0 ;
                                                        backupAssigned = 0 ;
                                                        waitBackupReply = 1 ;
                                                        sendHelloBackup = 0 ;
                                                        set( backupDelay, 15 )
                                                :: else
                                                fi ;

                                                if
                                                :: ranklistEmpty == 1 ->
                                                        sendMessage( BackupAssign, Unicast, 0, id,
src ) ;

                                                        lastBackup = src ;
                                                        backupRank = msg ;
                                                        assignBackup = 0 ;
                                                        backupAssigned = 0 ;
                                                        waitBackupReply = 1 ;
                                                        sendHelloBackup = 0 ;
                                                        set( backupDelay, 15 ) ;
                                                        ranklistEmpty = 0
                                                :: else
                                                fi ;

                                                updateRanklist( msg, src )

                                        :: msgType == SrvSearch ->
                                                /*sendMessage( SrvSearchReply, Unicast, 0, id, src )
;*/
                                                checkRanklist( src )

                                        :: else -> checkRanklist( src )
                                        fi
                                fi ;

                                clearMail()
                        }

                /*** Announcements ***/
                :: centralNego == 0 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;
                        set( waitDelay, 30 )

                /*** Backup Assignment ***/
```

```
:: centralNego == 0 && assignBackup == 1 ->
        delay( nodeDelay, 1 ) ;
        findHighest() ;

        if
        :: anID == 255 ->
                printf( "MSC: Empty ranklist\n" ) ;
                ranklistEmpty = 1 ;
                assignBackup = 0 ;
                backupAssigned = 0 ;
                waitBackupReply = 0 ;
                lastBackup = 255 ;
                backupRank = 0 ;
                sendHelloBackup = 0
        :: else ->
                lastBackup = anID ;
                backupRank = aRank ;
                sendMessage( BackupAssign, Unicast, 0, id, anID ) ;
                assignBackup = 0 ;
                backupAssigned = 0 ;
                waitBackupReply = 1 ;
                sendHelloBackup = 0 ;
                set( backupDelay, 30 ) ;
        fi

:: waitBackupReply == 1 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( BackupAssign, Unicast, 0, id, lastBackup ) ;
        assignBackup = 0 ;
        backupAssigned = 0 ;
        waitBackupReply = 2 ;
        sendHelloBackup = 0 ;
        set( backupDelay, 30 ) ;

:: waitBackupReply == 2 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        printf("MSC: Remove\n") ;
        removeEntry( lastBackup ) ;
        lastBackup = 255 ;
        backupRank = 0 ;
        assignBackup = 1 ;
        backupAssigned = 0 ;
        waitBackupReply = 0 ;
        sendHelloBackup = 0 ;

/*** Backup polling ***/
:: sendHelloBackup == 1 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
        sendHelloBackup = 2 ;
        set( backupDelay, 30 )
:: sendHelloBackup == 2 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
        sendHelloBackup = 3 ;
        set( backupDelay, 30 )
```

```
                :: sendHelloBackup == 3 && backupDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        removeEntry( lastBackup ) ;
                        lastBackup = 255 ;
                        backupRank = 0 ;
                        assignBackup = 1 ;
                        backupAssigned = 0 ;
                        waitBackupReply = 0 ;
                        sendHelloBackup = 0 ;

                /*** Central negotation ***/
                :: centralNego == 1 && centralNegoMax.val <= 0 ->
                        if
                        :: from_mail_to_i ?? [eval(IAmCentral), transType, msg, src, dest] ||
                                from_mail_to_i ?? [eval(CentralNego), transType, msg, src, dest] ->
                                        set( centralNegoMax, 30 )
                        :: else ->
                                centralNego = 0 ;
                                reset( waitDelay ) ;

                                do
                                :: my_nego_list ? _, _, _
                                :: empty( my_nego_list ) -> break
                                od
                        fi

                /*** Central appointment ***/
                :: waitAppointAck == 1 && centralAppointDelay.val <= 0 ->
                        sendMessage( NewCentralAssign, Unicast, rank, id, lastAppointed ) ;
                        waitAppointAck = 2 ;
                        set( centralAppointDelay, 30 )
                :: waitAppointAck == 2 && centralAppointDelay.val <= 0 ->
                        waitAppointAck = 0 ;
                        lastAppointed = 255
                od ;

        :: mode == 3 ->
                printf("MSC: NORMAL\n") ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: msgType == IAmCentral ->
                                        lastCentral = src ;

                                :: msgType == BackupAssign ->
                                        if
                                        :: lastCentral != 255 && src != lastCentral ->
                                                sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
                                                sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
```

```
                                        :: else ->
                                                sendMessage( BackupAssignAck, Unicast, 0, id, src )
;

                                                clearVariables() ;
                                                lastCentral = src ;

                                                mode = 4 ;
                                                break
                                        fi

                                :: msgType == NewCentralAssign ->
                                        sendMessage( CentralAssignAck, Unicast, 0, id, src ) ;

                                        clearVariables() ;
                                        lastBackup = src ;
                                        backupRank = msg ;
                                        sendHelloBackup = 1 ;
                                        assignBackup = 0 ;
                                        backupAssigned = 1 ;

                                        mode = 2 ;
                                        break

                                :: msgType == MyResRqst ->
                                        sendMessage( MyResReply, Unicast, rank, id, src ) ;

                                :: else
                                fi ;

                                clearMail()
                        }

                /*** Periodic searching ***/
                :: lastCentral != 255 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( SrvSearch, Unicast, 0, id, lastCentral ) ;
                        set( waitDelay, 30 )
                od

        :: mode == 4 ->
                printf("MSC: BACKUP\n") ;
                nr_backup++ ;
                set( waitDelay, 60 ) ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
//                              :: msgType == IAmCentral ->
//                                      if
//                                      :: src != lastCentral ->
```

```
//                                              sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
//                                              sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
//                                      :: else
//                                      fi

                                    :: msgType == BackupAssign ->
                                            if
                                            :: src == lastCentral ->
                                                    sendMessage( BackupAssignAck, Unicast, 0, id,
lastCentral )
                                            :: else ->
                                                    sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
                                                    sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                            fi

                                    :: msgType == BackupCancel ->
                                            clearRanklist() ;
                                            clearVariables() ;

                                            mode = 3 ;
                                            nr_backup-- ;
                                            break

                                    :: msgType == HelloDevice ->
                                            if
                                            :: src == lastCentral ->
                                                    sendMessage( HelloCentral, Unicast, 0, id,
lastCentral ) ;

                                                    sendHelloCentral = 0 ;
                                                    set( waitDelay, 60 )
                                            :: else //->
//                                                  sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
//                                                  sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                            fi

                                    :: else
                                    fi ;

                                    clearMail()
                            }

                    /*** Central polling ***/
                    :: sendHelloCentral == 0 && waitDelay.val <= 0 ->
                            delay( nodeDelay, 1 ) ;
                            sendMessage( HelloCentral, Unicast, 0, id, lastCentral ) ;
                            sendHelloCentral = 1 ;
                            set( waitDelay, 60 )
                    :: sendHelloCentral == 1 && waitDelay.val <= 0 ->
                            delay( nodeDelay, 1 ) ;
```

```
                        clearVariables() ;

                        mode = 2 ;
                        nr_backup-- ;
                        break
                od
        od
}

init
{
        atomic
        {
                run Node(0, 10, 2) ;
                run Node(1, 4, 3) ;
                run Node(2, 8, 3)
        }
}



//Backup Takeover with Central Failure (no message loss)
// Ceryen Tan, Vasughi Sundramoorthy
// 8/20 - Clear buffer error
// 8/17 - Message renaming
// 8/16 - Cleanup attempt

#include "dtime.h"

#define N 3
#define MAX_CH 6

#define Unicast 0
#define Broadcast 1

mtype = { BackupAssign, BackupAssignAck, HelloCentral, HelloDevice } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;

byte nr_leader = 0 ;
byte nr_backup = 0 ;
byte nr_poll = 0 ;
bit reset = 0 ;

inline clearBuffer()
{
        atomic
        {
                do
                :: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
                :: empty(from_mail_to_i) -> break
                od
        }
}

inline clearVariables()
```

```
{
	lastCentral = 255 ;
	lastBackup = 255 ;
	sendHelloBackup = 0 ;
	sendHelloCentral = 0 ;

	assignBackup = 0 ;
	backupAssigned = 0 ;
	waitBackupReply = 0 ;

	reset( waitDelay ) ;
	reset( nodeDelay ) ;
	reset( backupDelay )
}

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
	atomic
	{
		if
		:: transType == Unicast -> from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
		:: transType == Broadcast ->
			dest = 0 ;

			do
			:: dest < N ->
				if
				:: dest != id -> from_mail[dest] ! msgType, transType, aMsg, aSrc, dest
;
				:: else
				fi ;

				dest++
			:: dest >= N -> dest = 0 ; break
			od ;
		fi
	}
}

proctype Node(byte id, rank, mode, nextInRank)
{
	/* Mail */
	chan from_mail_to_i = from_mail[id] ;
	xr from_mail_to_i ;

	/* Messages */
	mtype msgType ;
```

```
	byte transType ;
	byte msg ;
	byte src ;
	byte dest ;

	/* Backup */
	byte lastCentral ;
	byte lastBackup ;
	byte sendHelloBackup ;
	byte sendHelloCentral ;

	bit assignBackup ;
	bit backupAssigned ;
	byte waitBackupReply ;

	/* Timers */
	timer waitDelay ;
	timer nodeDelay ;
	timer backupDelay ;

	printf("MSC: START\n") ;
	atomic { clearBuffer() } ;

start:
	atomic { clearVariables() } ;

	do
	:: mode == 2 ->                                 /***** Central mode *****/
		nr_leader++ ;
		printf("MSC: CENTRAL\n");

		clearVariables() ;
		assignBackup = 1 ;

		do
		:: from_mail_to_i ? msgType, transType, msg, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: transType == Broadcast -> skip

				:: transType == Unicast ->
					if
					:: msgType == BackupAssignAck ->
						if
						:: src == lastBackup ->
							assignBackup = 0 ;
							backupAssigned = 1 ;
							waitBackupReply = 0 ;
							sendHelloBackup = 1 ;
							set( backupDelay, 15 )
						:: else
						fi
```

```
                        :: msgType == HelloCentral ->
                                if
                                :: src == lastBackup ->
                                        sendHelloBackup = 1 ;

                                        if
                                        :: reset == 0 ->
                                                nr_poll++ ;

                                                if
                                                :: nr_poll == 4 ->
                                                        nr_poll = 0 ;
                                                        reset = 1 ;

                                                        clearVariables() ;

                                                        nr_leader-- ;
                                                        goto down
                                                :: else
                                                fi
                                        :: else
                                        fi ;

                                        set( backupDelay, 15 )
                                :: else
                                fi

                        :: else
                        fi
                fi ;

                clearMail()
        }

/*** Backup assignment ***/
:: assignBackup == 1 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( BackupAssign, Unicast, 0, id, nextInRank ) ;
        lastBackup = nextInRank ;
        assignBackup = 0 ;
        waitBackupReply = 1 ;
        set( backupDelay, 15 )

:: waitBackupReply == 1 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        waitBackupReply = 0 ;
        lastBackup = 255 ;
        assignBackup = 1

/*** Backup polling ***/
:: sendHelloBackup == 1 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
        sendHelloBackup = 2 ;
        set( backupDelay, 15 )
:: sendHelloBackup == 2 && backupDelay.val <= 0 ->
```

```
                delay( nodeDelay, 1 ) ;
                sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
                sendHelloBackup = 3 ;
                set( backupDelay, 15 )
        :: sendHelloBackup == 3 && backupDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                lastBackup = 255 ;
                assignBackup = 1 ;
                backupAssigned = 0
        od

:: mode == 3 ->                         /***** Normal mode *****/
        clearVariables() ;

        do
        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                delay( nodeDelay, 1 ) ;

                atomic
                {
                        if
                        :: transType == Broadcast -> skip

                        :: transType == Unicast ->
                                if
                                :: msgType == BackupAssign ->
                                        sendMessage( BackupAssignAck, Unicast, 0, id, src )
;

                                        mode = 4 ;
                                        lastCentral = src ;
                                        break

                                :: else
                                fi
                        fi ;

                        clearMail()
                }

        :: empty(from_mail_to_i) -> delay( nodeDelay, 1)
        od

:: mode == 4 ->
        nr_backup++ ;
        printf("MSC: BACKUP\n") ;

        sendHelloCentral = 0 ;
        set( waitDelay, 30 ) ;

        do
        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                delay( nodeDelay, 1 ) ;

                atomic
                {
```

```
                                        if
                                        :: transType == Broadcast -> skip

                                        :: transType == Unicast ->
                                                if
                                                :: msgType == HelloDevice ->
                                                        if
                                                        :: src == lastCentral ->
                                                                sendMessage( HelloCentral, Unicast, 0, id,
src ) ;

                                                                sendHelloCentral = 1 ;
                                                                set( waitDelay, 30 )
                                                        :: else
                                                        fi

                                                :: else
                                                fi
                                        fi ;

                                        clearMail()
                                }

                        /*** Central polling ***/
                        :: sendHelloCentral == 0 && waitDelay.val <= 0 ->               /* MAKE SURE VALID
*/
                                delay( nodeDelay, 1 ) ;
                                sendMessage( HelloCentral, Unicast, 0, id, lastCentral ) ;
                                sendHelloCentral = 2 ;
                                set( waitDelay, 30 )
                        :: sendHelloCentral == 1 && waitDelay.val <= 0 ->
                                delay( nodeDelay, 1 ) ;
                                sendMessage( HelloCentral, Unicast, 0, id, lastCentral ) ;
                                sendHelloCentral = 2 ;
                                set( waitDelay, 30 )
                        :: sendHelloCentral == 2 && waitDelay.val <= 0 ->
                                delay( nodeDelay, 1 ) ;
                                mode = 2 ;
                                nr_backup -- ;

                                clearVariables() ;

                                break
                        od
        od ;

down:
        printf("MSC: DOWN\n") ;
        do
        :: from_mail_to_i ? _, _, _, _, _
        od
}

init
{
        atomic
```

```
	{
		run Node(0, 10, 2, 2) ;
		run Node(1, 4, 3, 255) ;
		run Node(2, 8, 3, 1)
	}
}



// Leader election with Periodic Announcements
// Ceryen Tan, Vasughi Sundramoorthy
// 8/20 - Clear buffer error
// 8/17 - Message renaming
// 8/16 - Cleanup attempt

#include "dtime.h"

#define N 3
#define MAX_CH 10
#define MAX_LOSS 4

#define Unicast 0
#define Broadcast 1

mtype = { IAmCentral, CentralNego, LeaderElect } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
chan nego_list[N] = [N] of {byte, bit, bit } ;

byte nr_leader = 0 ;
byte rightAssigned = 0 ;

byte lossCounters[3] ;

inline clearBuffer()
{
	do
	:: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
	:: empty(from_mail_to_i) -> break
	od
}

inline clearVariables()
{
	announceReceived = 0 ;
	negoReceived = 0 ;

	centralNego = 0 ;

	waitAnnounce = 0 ;

	reset( waitDelay ) ;
	reset( nodeDelay ) ;
	reset( centralNegoMax ) ;

	do
	:: nempty( my_nego_list ) -> my_nego_list ? _, _, _
```

```
		:: empty( my_nego_list ) -> break
		od
}

inline clearMail()
{
		msgType = 0 ;
		transType = 0 ;
		msg = 0 ;
		src = 0 ;
		dest = 0
}

inline getType( msgType )
{
		byte type ;

		if
		:: msgType == IAmCentral -> type = 0
		:: msgType == CentralNego -> type = 1
		:: msgType == LeaderElect -> type = 2
		:: else -> printf("MSC: ERROR\n")
		fi
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
		getType( msgType ) ;

		atomic
		{
			if
			:: transType == Unicast ->
				if
				:: type == 255 || lossCounters[type] >= MAX_LOSS ->
					from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
				:: else ->
					if
					:: from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
					:: true -> printf( "MSC: UNICAST: %d, %d, %d\n", aMsg, aSrc, dest )
; lossCounters[type]++
					fi
				fi ;
			:: transType == Broadcast ->
				dest = 0 ;

				do
				:: dest < N ->
					if
					:: dest != id ->
						if
						:: type == 255 || lossCounters[type] >= MAX_LOSS ->
							from_mail[dest] ! msgType, transType, aMsg, aSrc,
dest
						:: else ->
							if
```

```
                                                        :: from_mail[dest] ! msgType, transType, aMsg,
aSrc, dest
                                                        :: true -> printf( "MSC: BROADCAST: %d, %d,
%d\n", aMsg, aSrc, dest ) ;
                                                                lossCounters[type]++
                                                        fi
                                                fi ;
                                        :: else
                                        fi ;

                                        dest++
                                :: dest >= N -> dest = 0 ; break
                                od ;
                        fi ;

                        type = 0
                }
}

proctype Node(byte id, rank)
{
        /* Mail */
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* Central negotiation table */
        chan my_nego_list = nego_list[id] ;
        xr my_nego_list ;
        xs my_nego_list ;

        bit announceReceived ;
        bit negoReceived ;

        /* Messages */
        mtype msgType ;
        bit transType ;
        byte msg ;
        byte src ;
        byte dest ;

        /* Central flags */
        bit centralNego ;

        /* Normal flags */
        byte waitAnnounce ;

        /* Timers */
        timer waitDelay ;
        timer nodeDelay ;
        timer centralNegoMax ;

        printf("MSC: START\n") ;
        atomic { clearBuffer() } ;

start:
        atomic { clearVariables() } ;
```

```
central:
	nr_leader++ ;

	if
	:: id == 0 -> rightAssigned = 1
	:: else
	fi ;

	do
	:: from_mail_to_i ? msgType, transType, msg, src, dest ->
		delay( nodeDelay, 1 ) ;

		atomic
		{
			if
			:: src != id ->
				if
				:: msgType == IAmCentral ->
					announceReceived = 0 ;
					negoReceived = 0 ;

					if
					:: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
						my_nego_list ?? src, announceReceived,
negoReceived
					:: else
					fi ;

					if
					:: !(announceReceived == 1 && negoReceived == 1) ->
						sendMessage( CentralNego, Unicast, rank, id, src ) ;
						centralNego = 1 ;
						reset( waitDelay ) ;
						set( centralNegoMax, 30 ) ;
					:: else
					fi ;

					announceReceived = 1 ;
					negoReceived = 0 ;
					my_nego_list ! src, announceReceived, negoReceived ;

					announceReceived = 0 ;
					negoReceived = 0

				:: msgType == CentralNego ->
					announceReceived = 0 ;
					negoReceived = 0 ;

					if
					:: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
						my_nego_list ?? src, announceReceived,
negoReceived
					:: else
```

```
                                        fi ;

                                        if
                                        :: ( announceReceived == 1 && negoReceived == 0 ) || (
announceReceived == 0 ) ->
                                                sendMessage( CentralNego, Unicast, rank, id, src ) ;
                                        :: else
                                        fi ;

                                        negoReceived = 1 ;
                                        my_nego_list ! src, announceReceived, negoReceived ;

                                        announceReceived = 0 ;
                                        negoReceived = 0 ;

                                        if
                                        :: msg > rank ->
                                                printf("MSC: LOST\n") ;

                                                clearVariables() ;
                                                waitAnnounce = 1 ;

                                                if
                                                :: id == 0 -> rightAssigned = 0
                                                :: else
                                                fi ;

                                                nr_leader-- ;
                                                goto normal

                                        :: else ->
                                                if
                                                :: centralNego == 0 ->
                                                        centralNego = 1 ;
                                                        reset( waitDelay ) ;
                                                        set( centralNegoMax, 30 )
                                                :: else
                                                fi
                                        fi

                                :: msgType == LeaderElect ->
                                        sendMessage( IAmCentral, Broadcast, 0, id, 255 )

                                        :: else
                                        fi
                        :: else
                        fi ;

                        clearMail()
                }

        /*** Announcements ***/
        :: centralNego == 0 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;
                set( waitDelay, 30 )
```

```
		/*** Central negotiation ***/
		:: centralNego == 1 && centralNegoMax.val <= 0 ->
			atomic
			{
				if
				:: from_mail_to_i ?? [eval(IAmCentral), transType, msg, src, dest] ||
					from_mail_to_i ?? [eval(CentralNego), transType, msg, src, dest] ->
						set( centralNegoMax, 30 )
				:: else ->
					centralNego = 0 ;

					do
					:: my_nego_list ? _, _, _
					:: empty( my_nego_list ) -> break
					od
				fi
			}
		od ;

normal:
		set( waitDelay, 60 ) ;

		do
		:: from_mail_to_i ? msgType, transType, msg, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: msgType == IAmCentral ->
					waitAnnounce = 0 ;
					reset( waitDelay )

				:: else
				fi ;

				clearMail()
			}

		/*** Initial wait for announcement ***/
		:: waitAnnounce == 1 && waitDelay.val <= 0 ->
			delay( nodeDelay, 1 ) ;
			sendMessage( LeaderElect, Broadcast, 0, id, 255 ) ;
			waitAnnounce = 2 ;
			set( waitDelay, 30 ) ;

		:: waitAnnounce == 2 && waitDelay.val <= 0 ->
			delay( nodeDelay, 1 ) ;
			goto central
		od ;
}

init
{
		atomic
```

```
	{
		run Node(0, 10) ;
		run Node(1, 4) ;
		run Node(2, 6)
	}
}



// Leader Election Request with message loss
// Ceryen Tan, Vasughi Sundramoorthy
// 8/20 - Clear buffer error
// 8/17 - Cleanup attempt, Message renaming

#include "dtime.h"

#define N 3
#define MAX_CH 10
#define MAX_LOSS 1

#define Unicast 0
#define Broadcast 1

mtype =
	{
		MyResource, IAmCentral, CentralNego, LeaderElect,
		NewCentralAssign, CentralAssignAck,
		SrvRegRqst, SrvReg, UnsolSrvReg, UnsolSrvRegAck
	} ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
chan nego_list[N] = [N] of {byte, bit, bit } ;

byte nr_leader = 0 ;
byte rightAssigned = 0 ;

byte lossCounters[10] ;

inline clearBuffer()
{
	atomic
	{
		do
		:: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
		:: empty(from_mail_to_i) -> break
		od
	}
}

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
```

```
	dest = 0
}

inline clearVariables()
{
	announceReceived = 0 ;
	negoReceived = 0 ;

	mode = 0 ;
	waitMode = 0 ;
	centralNego = 0 ;
	setRegister = 0 ;

	waitAppointAck = 0 ;
	lastAppointed = 255 ;

	lastCentral = 255 ;
	waitAnnounce = 0 ;
	waitSrvRqst = 0 ;
	waitSrvRegReply = 0 ;

	reset( waitDelay ) ;
	reset( nodeDelay ) ;
	reset( centralNegoMax ) ;
	reset( centralAppointDelay ) ;

	do
	:: my_nego_list ? _, _, _
	:: empty( my_nego_list ) -> break
	od
}

inline getType( msgType )
{
	byte type ;

	if
	:: msgType == MyResource -> type = 0
	:: msgType == IAmCentral -> type = 1
	:: msgType == CentralNego -> type = 2
	:: msgType == LeaderElect -> type = 3
	:: msgType == NewCentralAssign -> type = 4
	:: msgType == SrvRegRqst -> type = 5
	:: msgType == SrvReg -> type = 6
	:: msgType == UnsolSrvReg -> type = 7
	:: msgType == UnsolSrvRegAck -> type = 8
	:: msgType == CentralAssignAck -> type = 9
	:: else -> type = 255
	fi
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
	getType( msgType ) ;

	atomic
```

```
	{
		if
		:: transType == Unicast ->
			if
			:: type == 255 || lossCounters[type] >= MAX_LOSS ->
				from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
			:: else ->
				if
				:: from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
				:: true -> printf( "MSC: UNICAST: %d, %d, %d\n", aMsg, aSrc, dest )
; lossCounters[type]++
				fi
			fi ;
		:: transType == Broadcast ->
			dest = 0 ;

			do
			:: dest < N ->
				if
				:: dest != id ->
					if
					:: type == 255 || lossCounters[type] >= MAX_LOSS ->
						from_mail[dest] ! msgType, transType, aMsg, aSrc,
dest
					:: else ->
						if
						:: from_mail[dest] ! msgType, transType, aMsg,
aSrc, dest
						:: true -> printf( "MSC: BROADCAST: %d, %d,
%d\n", aMsg, aSrc, dest ) ;
							lossCounters[type]++
						fi
					fi ;
				:: else
				fi ;

				dest++
			:: dest >= N -> dest = 0 ; break
			od ;
		fi ;

		type = 0
	}
}

proctype Node(byte id, rank)
{
	/* Mail */
	chan from_mail_to_i = from_mail[id] ;
	xr from_mail_to_i ;

	/* Central negotiation table */
	chan my_nego_list = nego_list[id] ;
	xr my_nego_list ;
	xs my_nego_list ;
```

```
        bit announceReceived ;
        bit negoReceived ;

        /* Messages */
        mtype msgType ;
        bit transType ;
        byte msg ;
        byte src ;
        byte dest ;

        /* Central flags */
        byte mode ;
        byte waitMode ;
        bit centralNego ;
        bit setRegister ;

        /* Central appointment */
        byte waitAppointAck ;
        byte lastAppointed ;

        /* Normal flags */
        byte lastCentral ;
        byte waitAnnounce ;
        bit waitSrvRqst ;
        byte waitSrvRegReply ;

        /* Timers */
        timer waitDelay ;
        timer nodeDelay ;
        timer centralNegoMax ;
        timer centralAppointDelay ;

        printf("MSC: START\n") ;
        atomic { clearBuffer() } ;

start:
        atomic { clearVariables() } ;

        do
        :: mode == 0 ->                                 /***** Startup *****/
                sendMessage( MyResource, Broadcast, rank, id, 255 ) ;
                delay( nodeDelay, 1 ) ;
                nr_leader++ ;
                mode++

        :: mode == 1 ->                                 /***** Listen mode *****/
                set( waitDelay, 30 ) ;

                do
                :: waitDelay.val > 0 ->
                        if
                        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                                delay( nodeDelay, 1 ) ;

                                atomic
                                {
```

```
                                        if
                                        :: transType == Broadcast ->
                                                if
                                                :: src != id ->
                                                        if
                                                        :: msgType == MyResource ->
                                                                printf( "MSC: %d\n", msg ) ;

                                                                if
                                                                :: msg > rank ->
                                                                        printf( "MSC: LOST\n" ) ;

                                                                        clearVariables() ;
                                                                        waitAnnounce = 1 ;

                                                                        mode = 3 ;
                                                                        nr_leader-- ;
                                                                        break
                                                                :: else
                                                                fi

                                                        :: else
                                                        fi
                                                :: else
                                                fi

                                        :: transType == Unicast ->
                                                if
                                                :: msgType == NewCentralAssign ->
                                                        printf( "MSC: WON\n" ) ;
                                                        sendMessage( CentralAssignAck, Unicast,
0, id, src ) ;

                                                        clearVariables() ;

                                                        mode = 2 ;
                                                        break

                                                :: msgType == SrvRegRqst ->
                                                        printf( "MSC: LOST\n" ) ;
                                                        sendMessage( SrvReg, Unicast, 0, id, src ) ;

                                                        clearVariables() ;

                                                        mode = 3 ;
                                                        nr_leader-- ;
                                                        break

                                                :: else
                                                fi
                                        fi ;

                                        clearMail()
                                }

                        :: empty(from_mail_to_i) -> delay( nodeDelay, 1 )
```

```
                        fi ;

                :: waitDelay.val <= 0 ->
                        printf( "MSC: WON\n") ;
                        delay( nodeDelay, 1 ) ;
                        mode = 2 ;
                        setRegister = 1 ;
                        break
                od

        :: mode == 2 ->                         /***** Central mode *****/
                printf("MSC: CENTRAL\n");

                if
                :: id == 0 -> rightAssigned = 1
                :: else
                fi ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast ->
                                        if
                                        :: src != id ->
                                                if
                                                :: msgType == IAmCentral ->
                                                        announceReceived = 0 ;
                                                        negoReceived = 0 ;

                                                        if
                                                        :: my_nego_list ?? [eval(src),
announceReceived, negoReceived] ->
                                                                my_nego_list ?? src,
announceReceived, negoReceived
                                                        :: else
                                                        fi ;

                                                        if
                                                        :: !(announceReceived == 1 &&
negoReceived == 1) ->
                                                                sendMessage( CentralNego,
Unicast, rank, id, src ) ;
                                                                centralNego = 1 ;
                                                                waitMode = 3 ;
                                                                reset( waitDelay ) ;
                                                                set( centralNegoMax, 30 ) ;
                                                        :: else
                                                        fi ;

                                                        announceReceived = 1 ;
                                                        negoReceived = 0 ;
```

```
                                        my_nego_list ! src, announceReceived,
negoReceived ;

                                        announceReceived = 0 ;
                                        negoReceived = 0

                                :: msgType == LeaderElect ->
                                        sendMessage( IAmCentral, Broadcast, 0, id,
255 ) ;

                                :: msgType == MyResource ->
                                        if
                                        :: msg > rank ->
                                                sendMessage( NewCentralAssign,
Unicast, 0, id, src ) ;

                                                waitAppointAck = 1 ;
                                                lastAppointed = src ;
                                                set( centralAppointDelay, 30 )

                                        :: msg <= rank ->
                                                sendMessage( SrvRegRqst,
Unicast, 0, id, src ) ;
                                        fi

                                :: else
                                fi
                        :: else
                        fi

                :: transType == Unicast ->
                        if
                        :: msgType == CentralAssignAck ->
                                if
                                :: src == lastAppointed ->
                                        clearVariables() ;
                                        waitAnnounce = 1 ;

                                        if
                                        :: id == 0 -> rightAssigned = 0
                                        :: else
                                        fi ;

                                        mode = 3 ;
                                        nr_leader-- ;
                                        break
                                :: else
                                fi

                        :: msgType == CentralNego ->
                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
```

```
                                        my_nego_list ?? src, announceReceived,
negoReceived
                                    :: else
                                    fi ;

                                    if
                                    :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                        sendMessage( CentralNego, Unicast, rank,
id, src ) ;
                                    :: else
                                    fi ;

                                    negoReceived = 1 ;
                                    my_nego_list ! src, announceReceived,
negoReceived ;

                                    announceReceived = 0 ;
                                    negoReceived = 0 ;

                                    if
                                    :: msg > rank ->
                                        printf("MSC: LOST\n") ;

                                        clearVariables() ;
                                        waitAnnounce = 1 ;

                                        if
                                        :: id == 0 -> rightAssigned = 0
                                        :: else
                                        fi ;

                                        mode = 3 ;
                                        nr_leader-- ;
                                        break
                                    :: else ->
                                        if
                                        :: centralNego == 0 ->
                                            centralNego = 1 ;
                                            waitMode = 3 ;
                                            reset( waitDelay ) ;
                                            set( centralNegoMax, 30 )
                                        :: else
                                        fi
                                    fi

                            :: msgType == SrvReg -> skip

                            :: msgType == UnsolSrvReg ->
                                    sendMessage( UnsolSrvRegAck, Unicast, 0, id, src )

                            :: else
                            fi
                    fi ;

                    clearMail()
```

```
        }

/*** Announce modes ***/
:: waitMode == 0 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( IAmCentral, Broadcast, setRegister, id, 255 ) ;
        waitMode++ ;
        set( waitDelay, 10 )
:: waitMode == 1 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( IAmCentral, Broadcast, setRegister, id, 255 ) ;

        if
        :: setRegister == 1 -> setRegister = 0
        :: else -> waitMode++
        fi ;

        waitMode++ ;
        set( waitDelay, 10 )
:: waitMode == 2 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( SrvRegRqst, Broadcast, 0, id, 255 ) ;
        waitMode++ ;
        set( waitDelay, 30 )
:: centralNego == 0 && waitMode == 3 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;

        if
        :: setRegister == 1 -> setRegister = 0
        :: else -> waitMode++
        fi ;

        sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;
        set( waitDelay, 30 )

/*** Central Negotiation max timeout ***/
:: centralNego == 1 && centralNegoMax.val <= 0 ->
        if
        :: from_mail_to_i ?? [eval(IAmCentral), transType, msg, src, dest] ||
                from_mail_to_i ?? [eval(CentralNego), transType, msg, src, dest] ->
                        set( centralNegoMax, 30 )
        :: else ->
                centralNego = 0 ;
                waitMode = 3 ;
                reset( waitDelay ) ;

                do
                :: my_nego_list ? _, _, _
                :: empty( my_nego_list ) -> break
                od
        fi

/*** Central Appointment ***/
:: waitAppointAck == 1 && centralAppointDelay.val <= 0 ->
        sendMessage( NewCentralAssign, Unicast, 0, id, lastAppointed ) ;
        waitAppointAck = 2 ;
```

```
                    set( centralAppointDelay, 30 )
                :: waitAppointAck == 2 && centralAppointDelay.val <= 0 ->
                    waitAppointAck = 0 ;
                    lastAppointed = 255
                od

        :: mode == 3 ->                         /***** Normal mode *****/
                printf("MSC: NORMAL\n");
                if
                :: waitAnnounce == 1 -> set( waitDelay, 30 )
                :: waitSrvRqst == 1 -> set( waitDelay, 30 )
                :: waitSrvRegReply == 1 -> set( waitDelay, 30 )
                :: else
                fi ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                    delay( nodeDelay, 1 ) ;

                    atomic
                    {
                        if
                        :: transType == Broadcast ->
                            if
                            :: src != id ->
                                if
                                :: msgType == IAmCentral ->
                                    waitAnnounce = 0 ;

                                    if
                                    :: msg == 1 ->
                                        set( waitDelay, 30 ) ;
                                      printf("MSC: IamCentral, msg=1
Received\n") ;
                                        waitSrvRqst = 1 ;
                                        waitSrvRegReply = 0
                                    :: else ->
                                        if
                                        :: lastCentral != src || waitSrvRqst
== 1 ->
                                            sendMessage(
UnsolSrvReg, Unicast, 0, id, src ) ;
                                         printf("MSC: IamCentral,
msg=else Received\n") ;
                                            waitSrvRqst = 0 ;
                                            waitSrvRegReply = 1 ;
                                            set( waitDelay, 30 )
                                        :: else ->
                                            reset( waitDelay )
                                        fi
                                    fi ;

                                    lastCentral = src

                                :: msgType == LeaderElect ->
                                    waitAnnounce = 2 ;
```

```
                                        waitSrvRqst = 0 ;
                                        waitSrvRegReply = 0 ;
                                        set( waitDelay, 30 )

                                :: msgType == SrvRegRqst ->
                                        printf("MSC: SrvRegRqst Received\n") ;
                                        sendMessage( SrvReg, Unicast, 0, id, src ) ;

                                        waitAnnounce = 0 ;
                                        waitSrvRqst = 0 ;
                                        waitSrvRegReply = 0 ;

                                        lastCentral = src

                                :: else
                                fi
                        :: else
                        fi

                :: transType == Unicast ->
                        if
                        :: msgType == NewCentralAssign ->
                                clearVariables() ;

                                mode = 2 ;
                                nr_leader++ ;
                                break

                        :: msgType == UnsolSrvRegAck ->
                                if
                                :: waitSrvRegReply == 1 && src == lastCentral ->
                                        waitSrvRegReply = 0 ;
                                        reset( waitDelay ) ;
                                :: else
                                fi

                        :: else
                        fi
                fi ;

                clearMail()
        }

/*** Initial wait for announcement ***/
:: waitAnnounce == 1 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( LeaderElect, Broadcast, 0, id, 255 ) ;
        waitAnnounce = 2 ;
        waitSrvRqst = 0 ;
        waitSrvRegReply = 0 ;
        set( waitDelay, 30 )

/*** Wait for a service registration reply ***/
:: waitSrvRegReply == 1 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( LeaderElect, Broadcast, 0, id, 255 ) ;
```

```
                        lastCentral = 255 ;
                        waitAnnounce = 0 ;
                        waitSrvRqst = 0 ;
                        waitSrvRegReply = 2 ;
                        set( waitDelay, 30 )

                :: (waitAnnounce == 2 || waitSrvRegReply == 2) && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        goto start

                /*** Wait for a service registration request ***/
                :: waitSrvRqst == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( UnsolSrvReg, Unicast, 0, id, lastCentral ) ;

                        waitAnnounce = 0 ;
                        waitSrvRqst = 0 ;
                        waitSrvRegReply = 1 ;

                        set( waitDelay, 30 )
                od
        od
}

init
{
        atomic
        {
                run Node(0, 10) ;
                run Node(1, 4) ;
                run Node(2, 8)
        }
}


// Central Handover with no message loss
// Ceryen Tan, Vasughi Sundramoorthy
// 8/20 - Clear buffer error
// 8/17 - Message renaming
// 8/16 - Cleanup of code

#include "dtime.h"

#define N 3
#define MAX_CH 6

#define Unicast 0
#define Broadcast 1

mtype = {
                MyResource, IAmCentral,
                HelloCentral, HelloDevice, BackupCancel,
                NewCentralAssign, CentralAssignAck, RanklistTable
        } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
```

```
byte nr_leader = 0 ;
byte nr_backup = 0 ;
byte count = 0 ;
bit startup = 0 ;

inline clearBuffer()
{
	atomic
	{
		do
		:: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
		:: empty(from_mail_to_i) -> break
		od
	}
}

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0
}

inline clearVariables()
{
	lastCentral = 255 ;
	lastBackup = 255 ;
	sendHelloBackup = 0 ;
	sendHelloCentral = 0 ;

	waitAppointAck = 0 ;
	lastAppointed = 255 ;

	reset( waitDelay ) ;
	reset( nodeDelay ) ;
	reset( backupDelay ) ;
	reset( centralAppointDelay )
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
	atomic
	{
		if
		:: transType == Unicast -> from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
		:: transType == Broadcast ->
			dest = 0 ;

			do
			:: dest < N ->
				if
				:: dest != id -> from_mail[dest] ! msgType, transType, aMsg, aSrc, dest
;
				:: else
```

```
                                        fi ;

                                        dest++
                                :: dest >= N -> dest = 0 ; break
                                od ;
                        fi
                }
}

proctype Node( byte id, rank, mode )
{
        /* Mail */
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* Messages */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        /* Backup */
        byte lastCentral ;
        byte lastBackup ;
        byte sendHelloBackup ;
        bit sendHelloCentral ;

        /* Central Appoint */
        byte waitAppointAck ;
        byte lastAppointed ;

        /* Timers */
        timer waitDelay ;
        timer nodeDelay ;
        timer backupDelay ;
        timer centralAppointDelay ;

        printf("MSC: START\n") ;
        atomic { clearBuffer() } ;

start:
        atomic { clearVariables() } ;

        do
        :: mode == 1 ->
                do
                :: from_mail_to_i ? _, _, _, _, _ ->
                        if
                        :: startup == 1 -> break
                        :: else
                        fi
                :: startup == 1 -> break
                od ;

                delay( nodeDelay, 1 ) ;
```

```
                sendMessage( MyResource, Broadcast, rank, id, 255 ) ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast -> skip
                                :: transType == Unicast ->
                                        if
                                        :: msgType == NewCentralAssign ->
                                                sendMessage( CentralAssignAck, Unicast, 0, id, src )
;


                                                lastBackup = src ;
                                                sendHelloBackup = 1 ;

                                                mode = 2 ;
                                                break

                                        :: else
                                        fi
                                fi ;

                                clearMail()
                        }
                od

        :: mode == 2 ->
                printf("MSC: LEADER\n") ;

                if
                :: id == 0 -> sendHelloBackup = 1 ; lastBackup = 1
                :: else
                fi ;

                nr_leader++ ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast ->
                                        if
                                        :: src != id ->
                                                if
                                                :: msgType == MyResource ->
                                                        if
                                                        :: msg > rank ->
                                                                sendMessage( NewCentralAssign,
Unicast, rank, id, src ) ;
```

```
                                                waitAppointAck = 1 ;
                                                lastAppointed = src ;
                                                set( centralAppointDelay, 30 )

                                        :: else
                                        fi

                                :: else
                                fi
                        :: else
                        fi

                :: transType == Unicast ->
                        if
                        :: msgType == CentralAssignAck ->
                                if
                                :: src == lastAppointed ->
                                        if
                                        :: lastBackup != 255 ->
                                                sendMessage( BackupCancel,
Unicast, 0, id, lastBackup ) ;
                                        :: else
                                        fi ;

                                        sendMessage( RanklistTable, Unicast, 0, id,
src ) ;

                                        clearVariables() ;
                                        lastCentral = src ;

                                        mode = 4 ;
                                        nr_leader-- ;
                                        break
                                :: else
                                fi

                        :: msgType == HelloCentral ->
                                if
                                :: startup == 0 ->
                                        if
                                        :: count < 4 -> count++
                                        :: else -> count = 0 ; startup = 1
                                        fi
                                :: else
                                fi ;

                                if
                                :: src == lastBackup ->
                                        sendHelloBackup = 1 ;
                                        set( backupDelay, 15 )
                                :: else ->
                                        sendMessage( BackupCancel, Unicast, 0, id,
src )
                                fi
```

```
                                        :: else
                                        fi
                                fi ;

                                clearMail()
                        }

                /*** Announcements ***/
                :: waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;
                        set( waitDelay, 30 )

                /*** Backup polling ***/
                :: sendHelloBackup == 1 && backupDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
                        sendHelloBackup = 2 ;
                        set( backupDelay, 30 )
                :: sendHelloBackup == 2 && backupDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
                        sendHelloBackup = 3 ;
                        set( backupDelay, 30 )
                :: sendHelloBackup == 3 && backupDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        lastBackup = 255 ;
                        sendHelloBackup = 0

                /*** Central Appoint ***/
                :: waitAppointAck == 1 && centralAppointDelay.val <= 0 ->
                        sendMessage( NewCentralAssign, Unicast, rank, id, lastAppointed ) ;
                        waitAppointAck = 2 ;
                        set( centralAppointDelay, 30 )
                :: waitAppointAck == 2 && centralAppointDelay.val <= 0 ->
                        waitAppointAck = 0 ;
                        lastAppointed = 255
                od ;

:: mode == 3 ->
                printf("MSC: NORMAL\n") ;

                do
                :: from_mail_to_i ? _, _, _, _, _
                od

:: mode == 4 ->
                printf("MSC: BACKUP\n") ;
                nr_backup++ ;

                if
                :: id == 1 -> lastCentral = 0
                :: else
                fi ;

                sendHelloCentral = 0 ;
```

```
                set( waitDelay, 60 ) ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: msgType == HelloDevice ->
                                        if
                                        :: src == lastCentral ->
                                                sendMessage( HelloCentral, Unicast, 0, id,
lastCentral ) ;

                                                sendHelloCentral = 0 ;
                                                set( waitDelay, 60 )
                                        :: else
                                        fi

                                :: msgType == BackupCancel ->
                                        if
                                        :: src == lastCentral ->
                                                sendHelloCentral = 0 ;
                                                reset( waitDelay ) ;

                                                mode = 3 ;
                                                nr_backup-- ;
                                                break
                                        :: else
                                        fi

                                :: else
                                fi ;

                                clearMail()
                        }

                :: sendHelloCentral == 0 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( HelloCentral, Unicast, 0, id, lastCentral ) ;
                        sendHelloCentral = 1 ;
                        set( waitDelay, 60 )
                :: sendHelloCentral == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;

                        clearVariables() ;

                        mode = 2 ;
                        nr_backup-- ;
                        break
                od
        od
}

init
```

```
{
	atomic
	{
		run Node(0, 10, 2) ;
		run Node(1, 8, 4) ;
		run Node(2, 14, 1)
	}
}


// Registration for 300D with no message loss
// Ceryen Tan, Vasughi Sundramoorthy
// 8/17 - Cleanup attempt, message renaming

#include "dtime.h"

#define N 3
#define MAX_CH 5

#define Unicast 0
#define Broadcast 1

mtype =
	{
		IAmCentral,
		SrvRegRqst, SrvReg, UnsolSrvReg, SrvRegRenew,
		HelloDevice, HelloCentral,
		SrvFound, SrvNotFound, SrvSearch
	} ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;

bit found300D = 1;

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
	atomic
	{
		if
		:: transType == Unicast ->
			from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest

		:: transType == Broadcast ->
			dest = 0 ;

			do
			:: dest < N ->
				if
```

```
                                        :: dest != id -> from_mail[dest] ! msgType, transType, aMsg, aSrc, dest
;
                                        :: else
                                        fi ;

                                        dest++
                                :: dest >= N -> dest = 0 ; break
                                od ;
                        fi
                }
}

proctype Node300D( byte id, mode )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        byte lastCentral ;
        byte announceMode ;
        bit waitSrvRegRqst ;
        bit sendRegRenew ;

        bit reg300D ;

        bit wait300D ;

        timer regRenewDelay ;
        timer nodeDelay ;
        timer perAnnounceDelay ;
        timer waitDelay ;

        timer talk300D ;

        set( perAnnounceDelay, 60 ) ;

        if
        :: mode == 1 ->
                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast -> skip

                                :: transType == Unicast ->
                                        if
                                        :: msgType == SrvReg ->
```

```
                                        if
                                        :: msg == 1 ->
                                                printf("MSC: REG 300D\n");
                                                reg300D = 1 ;
                                                set( talk300D, 60 ) ;

                                                if
                                                :: wait300D == 1 -> sendMessage(
SrvFound, Unicast, 1, id, 2 ) ; wait300D = 0
                                                :: else
                                                fi
                                        :: else
                                        fi

                                :: msgType == SrvRegRenew ->
                                        if
                                        :: reg300D -> printf("MSC: 300D RENEW\n") ; set(
talk300D, 60 )
                                        :: else
                                        fi

                                :: msgType == SrvSearch ->
                                        if
                                        :: msg == 1 && reg300D == 1 -> sendMessage(
SrvFound, Unicast, 1, id, src )
                                        :: else -> sendMessage( SrvNotFound, Unicast, 0, id,
src ) ;

                                                if
                                                :: msg == 1 -> wait300D = 1
                                                :: else
                                                fi
                                        fi

                                :: msgType == UnsolSrvReg ->
                                        if
                                        :: msg == 1 ->
                                                printf("MSC: REG 300D\n");
                                                reg300D = 1 ;
                                                set( talk300D, 60 ) ;

                                                if
                                                :: wait300D == 1 -> sendMessage(
SrvFound, Unicast, 1, id, 2 ) ; wait300D = 0
                                                :: else
                                                fi
                                        :: else
                                        fi

                                :: else
                                fi
                        fi ;

                        clearMail()
                }
```

```
                /*** Announce modes ***/
                :: announceMode == 0 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( IAmCentral, Broadcast, 3, id, 255 ) ;
                        announceMode = 1 ;
                        set( waitDelay, 15 )

                :: announceMode == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( IAmCentral, Broadcast, 3, id, 255 ) ;
                        announceMode = 2 ;
                        set( waitDelay, 15 )

                :: announceMode == 2 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( SrvRegRqst, Unicast, 0, id, 1 ) ;
                        announceMode = 3

                /*** Periodic announcements ***/
                :: perAnnounceDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( IAmCentral, Broadcast, 2, id, 255 ) ;
                        set( perAnnounceDelay, 60 )

                /*** Renewal timeout ***/
                :: reg300D == 1 && talk300D.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        printf("MSC: UNREG 300D\n") ;
                        reg300D = 0
                od

        :: mode == 2 ->
                lastCentral = 255 ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast ->
                                        if
                                        :: msgType == IAmCentral ->
                                                if
                                                :: msg == 1 || msg == 2 ->
                                                        if
                                                        :: src != lastCentral ->
                                                                sendMessage( UnsolSrvReg,
Unicast, 1, id, src ) ;

                                                                sendRegRenew = 1 ;
                                                                set( regRenewDelay, 30 ) ;
                                                                lastCentral = src
                                                        :: else
                                                        fi
                                                :: msg == 3 ->
```

```
                                                lastCentral = src ;
                                                waitSrvRegRqst = 1 ;
                                                set( waitDelay, 30 )
                                        :: else
                                                fi
                                :: else
                                        fi
                        :: transType == Unicast ->
                                        if
                                        :: msgType == SrvRegRqst ->
                                                sendMessage( SrvReg, Unicast, 1, id, src )
                                                lastCentral = src ;
                                                sendRegRenew = 1 ;
                                                set( regRenewDelay, 30 ) ;
                                                waitSrvRegRqst = 0

                                        :: else
                                        fi
                        fi ;

                        clearMail()
                }

        /*** Registration renewal ***/
        :: sendRegRenew == 1 && regRenewDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( SrvRegRenew, Unicast, 0, id, lastCentral ) ;
                set( regRenewDelay, 30 )

        /*** Initial wait for service registration request ***/
        :: waitSrvRegRqst == 1 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( UnsolSrvReg, Unicast, 1, id, lastCentral ) ;
                waitSrvRegRqst = 0
        od
    fi
}

proctype Searcher( byte id, central )
{
    chan from_mail_to_i = from_mail[id] ;
    xr from_mail_to_i ;

    /* MESSAGES */
    mtype msgType ;
    byte transType ;
    byte msg ;
    byte src ;
    byte dest ;

    timer searchDelay ;
    timer nodeDelay ;

    do
    :: from_mail_to_i ? msgType, transType, msg, src, dest ->
            delay( nodeDelay, 1 ) ;
```

```
                atomic
                {
                        if
                        :: transType == Broadcast -> skip
                        :: transType == Unicast ->
                                if
                                :: msgType == SrvFound ->
                                        printf("MSC: FOUND\n") ;
                                        found300D = 1

                                :: msgType == SrvNotFound ->
                                        printf("MSC: NOT FOUND\n")

                                :: else
                                fi
                        fi ;

                        clearMail()
                }

        /*** Periodic search ***/
        :: searchDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;

                sendMessage( SrvSearch, Unicast, 1, id, central ) ;
                found300D = 0 ;

                set( searchDelay, 60 )
        od
}

init
{
        atomic
        {
                run Node300D( 0, 1 ) ;
                run Node300D( 1, 2 ) ;
                run Searcher( 2, 0 )
        }
}
```

// Registration for 3D with no message loss
// Ceryen Tan, Vasughi Sundramoorthy

```
// 8/17 - Cleanup attempt, Message renaming

#include "dtime.h"

#define N 3
#define MAX_CH 5

#define Unicast 0
#define Broadcast 1

mtype =
	{
		IAmCentral, SmallDeviceAnnounce,
		SrvRegRqst, SrvReg,
		HelloDevice, HelloCentral,
		SrvFound, SrvNotFound, SrvSearch
	} ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;

bit found3D = 1;

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
	atomic
	{
		if
		:: transType == Unicast ->
			from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest

		:: transType == Broadcast ->
			dest = 0 ;

			do
			:: dest < N ->
				if
				:: dest != id -> from_mail[dest] ! msgType, transType, aMsg, aSrc, dest
;
				:: else
				fi ;

				dest++
			:: dest >= N -> dest = 0 ; break
			od ;
		fi
	}
}
```

```
proctype Node300D( byte id )
{
	chan from_mail_to_i = from_mail[id] ;
	xr from_mail_to_i ;

	/* MESSAGES */
	mtype msgType ;
	byte transType ;
	byte msg ;
	byte src ;
	byte dest ;

	/* Central flags */
	bit reg3D ;
	bit wait3D ;
	byte wait3DHelloCentral ;

	/* Timers */
	timer nodeDelay ;
	timer perAnnounceDelay ;
	timer waitDelay ;
	timer talk3D ;

	set( perAnnounceDelay, 60 ) ;

	do
	:: from_mail_to_i ? msgType, transType, msg, src, dest ->
		delay( nodeDelay, 1 ) ;

		atomic
		{
			if
			:: transType == Broadcast ->
				if
				:: msgType == SmallDeviceAnnounce ->
					if
					:: reg3D == 0 -> sendMessage( SrvRegRqst, Unicast, 0, id, src
)
					:: else -> set( talk3D, 30 )
					fi

				:: else
				fi

			:: transType == Unicast ->
				if
				:: msgType == HelloCentral ->
					if
					:: src == 1 ->
						wait3DHelloCentral = 0 ;
						set( talk3D, 30 )
					:: else
					fi

				:: msgType == SmallDeviceAnnounce ->
					if
```

```
                                        :: reg3D == 0 -> sendMessage( SrvRegRqst, Unicast, 0, id, src
)
                                        :: else -> set( talk3D, 30 )
                                        fi

                                :: msgType == SrvReg ->
                                        if
                                        :: msg == 2 ->
                                                printf("MSC: REG 3D\n");
                                                reg3D = 1 ;
                                                wait3DHelloCentral = 0 ;
                                                set( talk3D, 30 ) ;

                                                if
                                                :: wait3D == 1 -> sendMessage( SrvFound, Unicast,
2, id, 2 ) ; wait3D = 0
                                                :: else
                                                fi
                                        :: else
                                        fi

                                :: msgType == SrvSearch ->
                                        if
                                        :: msg == 2 && reg3D == 1 -> sendMessage( SrvFound,
Unicast, 2, id, src )
                                        :: else -> sendMessage( SrvNotFound, Unicast, 0, id, src ) ;

                                                if
                                                :: msg == 2 -> wait3D = 1
                                                :: else
                                                fi
                                        fi

                                :: else
                                        fi
                        fi ;

                        clearMail()
                }

        /*** Periodic announcement ***/
        :: perAnnounceDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 2, id, 255 ) ;
                set( perAnnounceDelay, 60 )

        /*** 3D Polling ***/
        :: reg3D == 1 && wait3DHelloCentral == 0 && talk3D.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                printf("MSC: POLL 3D\n") ;
                sendMessage( HelloDevice, Unicast, 0, id, 1 ) ;
                wait3DHelloCentral = 1 ;
                set( talk3D, 30 )

        :: reg3D == 1 && wait3DHelloCentral == 1 && talk3D.val <= 0 ->
                delay( nodeDelay, 1 ) ;
```

```
                printf("MSC: POLL 3D\n") ;
                sendMessage( HelloDevice, Unicast, 0, id, 1 ) ;
                wait3DHelloCentral = 2 ;
                set( talk3D, 30 )

        :: reg3D == 1 && wait3DHelloCentral == 2 && talk3D.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                printf("MSC: UNREG 3D\n") ;
                reg3D = 0
        od
}

proctype Node3D( byte id )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        timer nodeDelay ;
        timer announceDelay ;
        timer perAnnounceDelay ;

        byte announceTime ;
        bit centralFound ;

        sendMessage( SmallDeviceAnnounce, Broadcast, 0, id, 255 ) ;
        announceTime = 5 ;
        set( announceDelay, announceTime ) ;
        set( perAnnounceDelay, 60 ) ;

        do
        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                delay( nodeDelay, 1 ) ;

                atomic
                {
                        if
                        :: transType == Broadcast ->
                                if
                                :: msgType == IAmCentral ->
                                        if
                                        :: msg >= 2 ->
                                                sendMessage( SmallDeviceAnnounce, Unicast, 0, id,
src )
                                        :: else
                                        fi ;
                                        centralFound = 1 ;
                                        announceTime = 0 ;
                                        reset( announceDelay )
                                :: else
```

```
                        fi

                    :: transType == Unicast ->
                        if
                        :: msgType == HelloDevice ->
                            sendMessage( HelloCentral, Unicast, 0, id, src )
                        :: msgType == SrvRegRqst ->
                            sendMessage( SrvReg, Unicast, 2, id, src ) ;
                            centralFound = 1 ;
                            reset( announceDelay )
                        :: else
                        fi
                    fi ;

                    clearMail()
                }

        /*** Initial exponential SmallDeviceAnnounce ***/
        :: centralFound == 0 && announceDelay.val <= 0 ->
            delay( nodeDelay, 1 ) ;
            sendMessage( SmallDeviceAnnounce, Broadcast, 0, id, 255 ) ;
            announceTime = announceTime * 2 ;

            if
            :: announceTime >= 30 -> announceTime = 5
            :: else
            fi ;

            set( announceDelay, announceTime )

        /*** Periodic SmallDeviceAnnounce ***/
        :: perAnnounceDelay.val <= 0 ->
            delay( nodeDelay, 1 ) ;
            sendMessage( SmallDeviceAnnounce, Broadcast, 0, id, 255 ) ;
            set( perAnnounceDelay, 60 )
        od
}

proctype Searcher( byte id, central )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        timer searchDelay ;
        timer nodeDelay ;

        do
        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
            delay( nodeDelay, 1 ) ;
```

```
                atomic
                {
                        if
                        :: transType == Broadcast -> skip
                        :: transType == Unicast ->
                                if
                                :: msgType == SrvFound ->
                                        printf("MSC: FOUND\n") ;
                                        found3D = 1

                                :: msgType == SrvNotFound ->
                                        printf("MSC: NOT FOUND\n")

                                :: else
                                fi
                        fi ;

                        clearMail()
                }

        /*** Periodic search ***/
        :: searchDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;

                sendMessage( SrvSearch, Unicast, 2, id, central ) ;
                found3D = 0 ;

                set( searchDelay, 60 )
        od
}

init
{
        atomic
        {
                run Node300D( 0 ) ;
                run Node3D( 1 ) ;
                run Searcher( 2, 0 )
        }
}
```

```
// Registration for 3D with message loss
// Ceryen Tan, Vasughi Sundramoorthy
```

```
// 8/17 - Cleanup attempt, Message renaming

#include "dtime.h"

#define N 3
#define MAX_CH 5
#define MAX_LOSS 1

#define Unicast 0
#define Broadcast 1

mtype =
	{
		IAmCentral, SmallDeviceAnnounce,
		SrvRegRqst, SrvReg,
		HelloDevice, HelloCentral,
		SrvFound, SrvNotFound, SrvSearch
	} ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;

bit found3D = 1;
byte lossCounters[9] ;

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0
}

inline getType( msgType )
{
	byte type ;

	if
	:: msgType == IAmCentral -> type = 0
	:: msgType == SrvRegRqst -> type = 1
	:: msgType == SrvReg -> type = 2
	:: msgType == SmallDeviceAnnounce -> type = 3
	:: msgType == HelloDevice -> type = 4
	:: msgType == HelloCentral -> type = 5
	:: msgType == SrvFound -> type = 6
	:: msgType == SrvNotFound -> type = 7
	:: msgType == SrvSearch -> type = 8
	:: else -> type = 255
	fi
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
	getType( msgType ) ;

	atomic
	{
```

```
                if
                :: transType == Unicast ->
                        if
                        :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                        :: else ->
                                if
                                :: from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                                :: true -> printf( "MSC: UNICAST: %d, %d, %d\n", aMsg, aSrc, dest )
; lossCounters[type]++
                                fi
                        fi ;
                :: transType == Broadcast ->
                        dest = 0 ;

                        do
                        :: dest < N ->
                                if
                                :: dest != id ->
                                        if
                                        :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                                from_mail[dest] ! msgType, transType, aMsg, aSrc,
dest
                                        :: else ->
                                                if
                                                :: from_mail[dest] ! msgType, transType, aMsg,
aSrc, dest
                                                :: true -> printf( "MSC: BROADCAST: %d, %d,
%d\n", aMsg, aSrc, dest ) ;
                                                        lossCounters[type]++
                                                fi
                                        fi ;
                                :: else
                                fi ;

                                dest++
                        :: dest >= N -> dest = 0 ; break
                        od ;
                fi ;

                type = 0
        }
}

proctype Node300D( byte id )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;
```

```
	/* Central flags */
	bit reg3D ;
	bit wait3D ;
	byte wait3DHelloCentral ;

	/* Timers */
	timer nodeDelay ;
	timer perAnnounceDelay ;
	timer waitDelay ;
	timer talk3D ;

	set( perAnnounceDelay, 60 ) ;

	do
	:: from_mail_to_i ? msgType, transType, msg, src, dest ->
		delay( nodeDelay, 1 ) ;

		atomic
		{
			if
			:: transType == Broadcast ->
				if
				:: msgType == SmallDeviceAnnounce ->
					if
					:: reg3D == 0 -> sendMessage( SrvRegRqst, Unicast, 0, id, src
)
					:: else -> set( talk3D, 30 )
					fi

				:: else
				fi

			:: transType == Unicast ->
				if
				:: msgType == HelloCentral ->
					if
					:: src == 1 ->
						wait3DHelloCentral = 0 ;
						set( talk3D, 30 )
					:: else
					fi

				:: msgType == SmallDeviceAnnounce ->
					if
					:: reg3D == 0 -> sendMessage( SrvRegRqst, Unicast, 0, id, src
)
					:: else -> set( talk3D, 30 )
					fi

				:: msgType == SrvReg ->
					if
					:: msg == 2 ->
						printf("MSC: REG 3D\n");
						reg3D = 1 ;
						wait3DHelloCentral = 0 ;
						set( talk3D, 30 ) ;
```

```
                                                    if
                                                    :: wait3D == 1 -> sendMessage( SrvFound, Unicast,
2, id, 2 ) ; wait3D = 0
                                                    :: else
                                                    fi
                                            :: else
                                            fi

                                    :: msgType == SrvSearch ->
                                            if
                                            :: msg == 2 && reg3D == 1 -> sendMessage( SrvFound,
Unicast, 2, id, src )
                                            :: else -> sendMessage( SrvNotFound, Unicast, 0, id, src ) ;

                                                    if
                                                    :: msg == 2 -> wait3D = 1
                                                    :: else
                                                    fi
                                            fi

                                    :: else
                                            fi
                            fi ;

                            clearMail()
                    }

            /*** Periodic announcement ***/
            :: perAnnounceDelay.val <= 0 ->
                    delay( nodeDelay, 1 ) ;
                    sendMessage( IAmCentral, Broadcast, 2, id, 255 ) ;
                    set( perAnnounceDelay, 60 )

            /*** 3D Polling ***/
            :: reg3D == 1 && wait3DHelloCentral == 0 && talk3D.val <= 0 ->
                    delay( nodeDelay, 1 ) ;
                    printf("MSC: POLL 3D\n") ;
                    sendMessage( HelloDevice, Unicast, 0, id, 1 ) ;
                    wait3DHelloCentral = 1 ;
                    set( talk3D, 30 )

            :: reg3D == 1 && wait3DHelloCentral == 1 && talk3D.val <= 0 ->
                    delay( nodeDelay, 1 ) ;
                    printf("MSC: POLL 3D\n") ;
                    sendMessage( HelloDevice, Unicast, 0, id, 1 ) ;
                    wait3DHelloCentral = 2 ;
                    set( talk3D, 30 )

            :: reg3D == 1 && wait3DHelloCentral == 2 && talk3D.val <= 0 ->
                    delay( nodeDelay, 1 ) ;
                    printf("MSC: UNREG 3D\n") ;
                    reg3D = 0
            od
}
```

```
proctype Node3D( byte id )
{
	chan from_mail_to_i = from_mail[id] ;
	xr from_mail_to_i ;

	/* MESSAGES */
	mtype msgType ;
	byte transType ;
	byte msg ;
	byte src ;
	byte dest ;

	timer nodeDelay ;
	timer announceDelay ;
	timer perAnnounceDelay ;

	byte announceTime ;
	bit centralFound ;

	sendMessage( SmallDeviceAnnounce, Broadcast, 0, id, 255 ) ;
	announceTime = 5 ;
	set( announceDelay, announceTime ) ;
	set( perAnnounceDelay, 60 ) ;

	do
	:: from_mail_to_i ? msgType, transType, msg, src, dest ->
		delay( nodeDelay, 1 ) ;

		atomic
		{
			if
			:: transType == Broadcast ->
				if
				:: msgType == IAmCentral ->
					if
					:: msg >= 2 ->
						sendMessage( SmallDeviceAnnounce, Unicast, 0, id,
src )
					:: else
					fi ;
					centralFound = 1 ;
					announceTime = 0 ;
					reset( announceDelay )
				:: else
				fi

			:: transType == Unicast ->
				if
				:: msgType == HelloDevice ->
					sendMessage( HelloCentral, Unicast, 0, id, src )
				:: msgType == SrvRegRqst ->
					sendMessage( SrvReg, Unicast, 2, id, src ) ;
					centralFound = 1 ;
					reset( announceDelay )
				:: else
				fi
```

```
                        fi ;

                        clearMail()
                }

        /*** Initial exponential SmallDeviceAnnounce ***/
        :: centralFound == 0 && announceDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( SmallDeviceAnnounce, Broadcast, 0, id, 255 ) ;
                announceTime = announceTime * 2 ;

                if
                :: announceTime >= 30 -> announceTime = 5
                :: else
                fi ;

                set( announceDelay, announceTime )

        /*** Periodic SmallDeviceAnnounce ***/
        :: perAnnounceDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( SmallDeviceAnnounce, Broadcast, 0, id, 255 ) ;
                set( perAnnounceDelay, 60 )
        od
}

proctype Searcher( byte id, central )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        timer searchDelay ;
        timer nodeDelay ;

        do
        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                delay( nodeDelay, 1 ) ;

                atomic
                {
                        if
                        :: transType == Broadcast -> skip
                        :: transType == Unicast ->
                                if
                                :: msgType == SrvFound ->
                                        printf("MSC: FOUND\n") ;
                                        found3D = 1

                                :: msgType == SrvNotFound ->
```

```
                                        printf("MSC: NOT FOUND\n")

                                    :: else
                                    fi
                            fi ;

                            clearMail()
                    }

            /*** Periodic search ***/
            :: searchDelay.val <= 0 ->
                    delay( nodeDelay, 1 ) ;

                    sendMessage( SrvSearch, Unicast, 2, id, central ) ;
                    found3D = 0 ;

                    set( searchDelay, 60 )
            od
}

init
{
            atomic
            {
                    run Node300D( 0 ) ;
                    run Node3D( 1 ) ;
                    run Searcher( 2, 0 )
            }
}


// Registration for 3C with no message loss
// Ceryen Tan, Vasughi Sundramoorthy
// 8/17 - Cleanup attempt, Message renaming

#include "dtime.h"

#define N 3
#define MAX_CH 5

#define Unicast 0
#define Broadcast 1

mtype =
            {
                    IAmCentral,
                    SrvReg,
                    HelloDevice, HelloCentral,
                    SrvFound, SrvNotFound, SrvSearch
            } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;

bit found3C = 1;

inline clearMail()
{
```

```
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
	atomic
	{
		if
		:: transType == Unicast ->
			from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest

		:: transType == Broadcast ->
			dest = 0 ;

			do
			:: dest < N ->
				if
				:: dest != id -> from_mail[dest] ! msgType, transType, aMsg, aSrc, dest
;
				:: else
				fi ;

				dest++
			:: dest >= N -> dest = 0 ; break
			od ;
		fi
	}
}

proctype Node300D( byte id, mode )
{
	chan from_mail_to_i = from_mail[id] ;
	xr from_mail_to_i ;

	/* MESSAGES */
	mtype msgType ;
	byte transType ;
	byte msg ;
	byte src ;
	byte dest ;

	byte lastCentral ;
	byte announceMode ;
	bit waitSrvRegRqst ;
	bit sendRegRenew ;

	bit reg3C ;
	bit wait3C ;
	byte wait3CHelloCentral ;

	timer nodeDelay ;
	timer perAnnounceDelay ;
```

```
timer waitDelay ;
timer talk3C ;

set( perAnnounceDelay, 60 ) ;

do
:: from_mail_to_i ? msgType, transType, msg, src, dest ->
        delay( nodeDelay, 1 ) ;

        atomic
        {
                if
                :: transType == Broadcast -> skip

                :: transType == Unicast ->
                        if
                        :: msgType == HelloCentral ->
                                if
                                :: src == 3 ->
                                        wait3CHelloCentral = 0 ;
                                        set( talk3C, 30 )
                                :: else
                                fi

                        :: msgType == SrvReg ->
                                if
                                :: msg == 3 ->
                                        printf("MSC: REG 3C\n");
                                        reg3C = 1 ;
                                        wait3CHelloCentral = 0 ;
                                        set( talk3C, 30 ) ;

                                        if
                                        :: wait3C == 1 -> sendMessage( SrvFound, Unicast,
3, id, 2 ) ; wait3C = 0
                                        :: else
                                        fi
                                :: else
                                fi

                        :: msgType == SrvSearch ->
                                if
                                :: msg == 3 && reg3C == 1 -> sendMessage( SrvFound,
Unicast, 3, id, src )
                                :: else -> sendMessage( SrvNotFound, Unicast, 0, id, src ) ;
                                        if
                                        :: msg == 3 -> wait3C = 1
                                        :: else
                                        fi
                                fi

                        :: else
                        fi
                fi ;

                clearMail()
```

```
                }

        /*** Announce modes ***/
        :: announceMode == 0 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 3, id, 255 ) ;
                announceMode = 1 ;
                set( waitDelay, 15 )

        :: announceMode == 1 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 3, id, 255 ) ;
                announceMode = 2

        /*** Periodic announce ***/
        :: perAnnounceDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 2, id, 255 ) ;
                set( perAnnounceDelay, 60 )

        /*** 3C Polling ***/
        :: reg3C == 1 && wait3CHelloCentral == 0 && talk3C.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                printf("MSC: POLL 3C\n") ;
                sendMessage( HelloDevice, Unicast, 0, id, 1 ) ;
                wait3CHelloCentral = 1 ;
                set( talk3C, 15 )

        :: reg3C == 1 && wait3CHelloCentral == 1 && talk3C.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                printf("MSC: POLL 3C\n") ;
                sendMessage( HelloDevice, Unicast, 0, id, 1 ) ;
                wait3CHelloCentral = 2 ;
                set( talk3C, 15 )

        :: reg3C == 1 && wait3CHelloCentral == 2 && talk3C.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                printf("MSC: UNREG 3C\n") ;
                reg3C = 0
        od
}

proctype Node3C( byte id )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        timer nodeDelay ;
```

```
	do
	:: from_mail_to_i ? msgType, transType, msg, src, dest ->
		delay( nodeDelay, 1 ) ;

		atomic
		{
			if
			:: transType == Broadcast ->
				if
				:: msgType == IAmCentral ->
					if
					:: msg >= 2 ->
						sendMessage( SrvReg, Unicast, 3, id, src )
					:: else
					fi

				:: else
				fi
			:: transType == Unicast ->
				if
				:: msgType == HelloDevice ->
					sendMessage( HelloCentral, Unicast, 0, id, src )

				:: else
				fi
			fi ;

			clearMail()
		}
	od
}

proctype Searcher( byte id, central )
{
	chan from_mail_to_i = from_mail[id] ;
	xr from_mail_to_i ;

	/* MESSAGES */
	mtype msgType ;
	byte transType ;
	byte msg ;
	byte src ;
	byte dest ;

	timer searchDelay ;
	timer nodeDelay ;

	do
	:: searchDelay.val <= 0 ->
		delay( nodeDelay, 1 ) ;

		sendMessage( SrvSearch, Unicast, 3, id, central ) ;
		found3C = 0 ;

		set( searchDelay, 60 )
```

```
		:: from_mail_to_i ? msgType, transType, msg, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: transType == Broadcast -> skip
				:: transType == Unicast ->
					if
					:: msgType == SrvFound ->
						printf("MSC: FOUND\n") ;
						found3C = 1

					:: msgType == SrvNotFound ->
						printf("MSC: NOT FOUND\n") ;
						delay( nodeDelay, 1 )

					:: else
					fi
				fi ;

				clearMail()
			}
		od
}

init
{
		atomic
		{
			run Node300D( 0, 1 ) ;
			run Node3C( 1 ) ;
			run Searcher( 2, 0 )
		}
}



// Registration for 3C with message loss
// Ceryen Tan, Vasughi Sundramoorthy
// 8/17 - Cleanup attempt, Message renaming

#include "dtime.h"

#define N 3
#define MAX_CH 5
#define MAX_LOSS 3

#define Unicast 0
#define Broadcast 1

mtype =
		{
			IAmCentral,
			SrvReg,
			HelloDevice, HelloCentral,
```

```
                SrvFound, SrvNotFound, SrvSearch
        } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;

bit found3C = 1;
byte lossCounters[7] ;

inline clearMail()
{
        msgType = 0 ;
        transType = 0 ;
        msg = 0 ;
        src = 0 ;
        dest = 0
}

inline getType( msgType )
{
        byte type ;

        if
        :: msgType == IAmCentral -> type = 0
        :: msgType == SrvReg -> type = 1
        :: msgType == HelloDevice -> type = 2
        :: msgType == HelloCentral -> type = 3
        :: msgType == SrvFound -> type = 4
        :: msgType == SrvNotFound -> type = 5
        :: msgType == SrvSearch -> type = 6
        :: else -> type = 255
        fi
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
        getType( msgType ) ;

        atomic
        {
                if
                :: transType == Unicast ->
                        if
                        :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                        :: else ->
                                if
                                :: from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                                :: true -> printf( "MSC: UNICAST: %d, %d, %d\n", aMsg, aSrc, dest )
; lossCounters[type]++
                                fi
                        fi ;
                :: transType == Broadcast ->
                        dest = 0 ;

                        do
                        :: dest < N ->
                                if
```

```
                                        :: dest != id ->
                                                if
                                                :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                                        from_mail[dest] ! msgType, transType, aMsg, aSrc,
dest
                                                :: else ->
                                                        if
                                                        :: from_mail[dest] ! msgType, transType, aMsg,
aSrc, dest
                                                        :: true -> printf( "MSC: BROADCAST: %d, %d,
%d\n", aMsg, aSrc, dest ) ;
                                                                lossCounters[type]++
                                                        fi
                                                fi ;
                                        :: else
                                        fi ;

                                        dest++
                                :: dest >= N -> dest = 0 ; break
                                od ;
                        fi ;

                        type = 0
                }
}

proctype Node300D( byte id, mode )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        byte announceMode ;

        bit reg3C ;
        bit wait3C ;
        byte wait3CHelloCentral ;

        timer nodeDelay ;
        timer perAnnounceDelay ;
        timer waitDelay ;
        timer talk3C ;

        set( perAnnounceDelay, 60 ) ;

        do
        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                delay( nodeDelay, 1 ) ;

                atomic
```

```
		{
			if
			:: transType == Broadcast -> skip

			:: transType == Unicast ->
				if
				:: msgType == HelloCentral ->
					if
					:: src == 3 ->
						wait3CHelloCentral = 0 ;
						set( talk3C, 30 )
					:: else
					fi

				:: msgType == SrvReg ->
					if
					:: msg == 3 ->
						printf("MSC: REG 3C\n");
						reg3C = 1 ;
						wait3CHelloCentral = 0 ;
						set( talk3C, 30 ) ;

						if
						:: wait3C == 1 -> sendMessage( SrvFound, Unicast,
3, id, 2 ) ; wait3C = 0
						:: else
						fi
					:: else
					fi

				:: msgType == SrvSearch ->
					if
					:: msg == 3 && reg3C == 1 -> sendMessage( SrvFound,
Unicast, 3, id, src )
					:: else -> sendMessage( SrvNotFound, Unicast, 0, id, src ) ;

						if
						:: msg == 3 -> wait3C = 1
						:: else
						fi
					fi

				:: else
				fi
			fi ;

			clearMail()
		}

	/*** Announce modes ***/
	:: announceMode == 0 && waitDelay.val <= 0 ->
		delay( nodeDelay, 1 ) ;
		sendMessage( IAmCentral, Broadcast, 3, id, 255 ) ;
		announceMode = 1 ;
		set( waitDelay, 15 )
```

```
		:: announceMode == 1 && waitDelay.val <= 0 ->
			delay( nodeDelay, 1 ) ;
			sendMessage( IAmCentral, Broadcast, 3, id, 255 ) ;
			announceMode = 2

		/*** Periodic announcement ***/
		:: perAnnounceDelay.val <= 0 ->
			delay( nodeDelay, 1 ) ;
			sendMessage( IAmCentral, Broadcast, 2, id, 255 ) ;
			set( perAnnounceDelay, 60 )

		/*** 3C Polling ***/
		:: reg3C == 1 && wait3CHelloCentral == 0 && talk3C.val <= 0 ->
			delay( nodeDelay, 1 ) ;
			printf("MSC: POLL 3C\n") ;
			sendMessage( HelloDevice, Unicast, 0, id, 1 ) ;
			wait3CHelloCentral = 1 ;
			set( talk3C, 15 )

		:: reg3C == 1 && wait3CHelloCentral == 1 && talk3C.val <= 0 ->
			delay( nodeDelay, 1 ) ;
			printf("MSC: POLL 3C\n") ;
			sendMessage( HelloDevice, Unicast, 0, id, 1 ) ;
			wait3CHelloCentral = 2 ;
			set( talk3C, 15 )

		:: reg3C == 1 && wait3CHelloCentral == 2 && talk3C.val <= 0 ->
			delay( nodeDelay, 1 ) ;
			printf("MSC: UNREG 3C\n") ;
			reg3C = 0
		od
}

proctype Node3C( byte id )
{
		chan from_mail_to_i = from_mail[id] ;
		xr from_mail_to_i ;

		/* MESSAGES */
		mtype msgType ;
		byte transType ;
		byte msg ;
		byte src ;
		byte dest ;

		timer nodeDelay ;

		do
		:: from_mail_to_i ? msgType, transType, msg, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: transType == Broadcast ->
					if
```

```
                                        :: msgType == IAmCentral ->
                                                if
                                                :: msg >= 2 ->
                                                        sendMessage( SrvReg, Unicast, 3, id, src )
                                                :: else
                                                fi

                                        :: else
                                        fi
                                :: transType == Unicast ->
                                        if
                                        :: msgType == HelloDevice ->
                                                sendMessage( HelloCentral, Unicast, 0, id, src )

                                        :: else
                                        fi
                                fi ;

                                clearMail()
                        }
                od
}

proctype Searcher( byte id, central )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        timer searchDelay ;
        timer nodeDelay ;

        do
        :: searchDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;

                sendMessage( SrvSearch, Unicast, 3, id, central ) ;
                found3C = 0 ;

                set( searchDelay, 60 )

        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                delay( nodeDelay, 1 ) ;

                atomic
                {
                        if
                        :: transType == Broadcast -> delay( nodeDelay, 1 )
                        :: transType == Unicast ->
                                if
```

```
                                        :: msgType == SrvFound ->
                                                printf("MSC: FOUND\n") ;
                                                found3C = 1

                                        :: msgType == SrvNotFound ->
                                                printf("MSC: NOT FOUND\n") ;

                                        :: else
                                        fi
                                fi ;

                                clearMail()
                        }
                od
}

init
{
        atomic
        {
                run Node300D( 0, 1 ) ;
                run Node3C( 1 ) ;
                run Searcher( 2, 0 )
        }
}



// 3-party Consistency Maintenance with no message loss
// Ceryen Tan, Vasughi Sundramoorthy
// 8/17 - Message renaming
// 8/16 - Cleanup attempt

#include "dtime.h"

#define N 4
#define MAX_CH 5

#define Unicast 0
#define Broadcast 1

mtype =
        {
                SubscriptionRqst, SubscriptionRqstAck, SubscriptionRenewal, Resubscribe,
                ServiceUpdate, ServiceUpdateRqst,
                HelloDevice, HelloCentral
        } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;

byte currentService ;
byte numUpdated ;

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
        atomic
        {
```

```
                if
                :: transType == Unicast -> from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;

                :: transType == Broadcast ->
                        dest = 0 ;

                        do
                        :: dest < N ->
                                if
                                :: dest != id -> from_mail[dest] ! msgType, transType, aMsg, aSrc, dest
                                :: else
                                fi ;

                                dest++
                        :: dest >= N -> dest = 0 ; break
                        od ;
                fi
        }
}

proctype Node300D( byte id, mode, recepient )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        /* Central flags */
        byte waitUpdate ;
        byte waitHello ;
        byte service ;

        bit subscribe1 ;
        bit subscribe2 ;
        timer renewal1 ;
        timer renewal2 ;
        timer talk1 ;
        timer talk2 ;

        /* Normal flags */
        bit sendSubRqst ;
        bit sendRenewal ;

        timer updateDelay ;

        timer nodeDelay ;
        timer waitDelay ;

        if
        :: mode == 1 ->
                sendMessage( ServiceUpdateRqst, Unicast, 0, id, recepient ) ;
```

```
		set( waitDelay, 60 ) ;
		waitUpdate = 1 ;

		do
		:: from_mail_to_i ? msgType, transType, msg, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: transType == Broadcast -> skip

				:: transType == Unicast ->
					if
					:: msgType == HelloCentral ->
						waitHello = 0 ;

						if
						:: waitUpdate == 2 ->
							sendMessage( ServiceUpdateRqst, Unicast,
0, id, recepient ) ;
							waitUpdate = 1 ;
							set( waitDelay, 60 )
						:: else
						fi

					:: msgType == ServiceUpdate ->
						service = msg ;
						waitUpdate = 1 ;
						waitHello = 0 ;
						set( waitDelay, 60 ) ;

						if
						:: subscribe1 == 1 ->
							sendMessage( ServiceUpdate, Unicast,
service, id, 1 ) ;
							set( talk1, 30 )
						:: else
						fi ;

						if
						:: subscribe2 == 1 ->
							sendMessage( ServiceUpdate, Unicast,
service, id, 2 ) ;
							set( talk2, 30 )
						:: else
						fi

					:: msgType == ServiceUpdateRqst ->
						sendMessage( ServiceUpdateRqst, Unicast, 0, id,
recepient )

					:: msgType == SubscriptionRenewal ->
						if
						:: src == 1 ->
							if
```

```
                                        :: subscribe1 == 1 -> set( renewal1, 90 )
                                        :: else -> sendMessage( Resubscribe,
Unicast, 0, id, src )
                                        fi
                                :: src == 2 ->
                                        if
                                        :: subscribe2 == 1 -> set( renewal2, 90 )
                                        :: else -> sendMessage( Resubscribe,
Unicast, 0, id, src )
                                        fi
                                fi

                        :: msgType == SubscriptionRqst ->
                                if
                                :: src == 1 -> subscribe1 = 1 ; set( renewal1, 90 )
                                :: src == 2 -> subscribe2 = 1 ; set( renewal2, 90 )
                                fi ;

                                sendMessage( SubscriptionRqstAck, Unicast, 0, id,
src ) ;
                                sendMessage( ServiceUpdateRqst, Unicast, 0, id,
recepient )
                                waitUpdate = 1 ;
                                set( waitDelay, 60 )

                        :: else
                        fi
                fi ;

                msgType = 0 ;
                transType = 0 ;
                msg = 0 ;
                src = 0 ;
                dest = 0
        }

:: waitUpdate == 1 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( HelloDevice, Unicast, 0, id, recepient ) ;
        waitUpdate = 2 ;
        waitHello = 1 ;
        set( waitDelay, 30 )

:: waitHello == 1 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( HelloDevice, Unicast, 0, id, recepient ) ;
        waitHello = 2 ;
        set( waitDelay, 30 )

:: waitHello == 2 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;                 /* NOW WHAT? */
        waitUpdate = 0 ;
        waitHello = 0 ;

:: subscribe1 == 1 && renewal1.val <= 0 ->
        delay( nodeDelay, 1 ) ;
```

```
                    subscribe1 = 0

                :: subscribe2 == 1 && renewal2.val <= 0 ->
                    delay( nodeDelay, 1 ) ;
                    subscribe2 = 0
                od

        :: mode == 2 ->
                sendMessage( SubscriptionRqst, Unicast, 0, id, recepient ) ;
                sendSubRqst = 1 ;
                sendRenewal = 0 ;
                waitUpdate = 0 ;
                set( waitDelay, 60 ) ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                    delay( nodeDelay, 1 ) ;

                    atomic
                    {
                        if
                        :: transType == Broadcast -> skip

                        :: transType == Unicast ->
                            if
                            :: msgType == ServiceUpdate ->
                                printf("MSC: %d\n", msg ) ;

                                if
                                :: service != msg && msg == currentService ->
numUpdated++
                                :: else
                                fi ;

                                service = msg ;
                                waitUpdate = 1 ;
                                set( updateDelay, 60 ) ;

                            :: msgType == Resubscribe ->
                                sendMessage( SubscriptionRqst, Unicast, 0, id,
recepient ) ;
                                sendSubRqst = 1 ;
                                sendRenewal = 0 ;
                                waitUpdate = 0 ;
                                reset( updateDelay ) ;
                                set( waitDelay, 60 )

                            :: msgType == SubscriptionRqstAck ->
                                sendSubRqst = 0 ;
                                sendRenewal = 1 ;
                                waitUpdate = 1 ;
                                set( updateDelay, 60 ) ;
                                set( waitDelay, 60 )

                            :: else
                            fi
```

```
                                        fi ;

                                        msgType = 0 ;
                                        transType = 0 ;
                                        msg = 0 ;
                                        src = 0 ;
                                        dest = 0
                                }

                        :: sendSubRqst == 1 && waitDelay.val <= 0 ->
                                delay( nodeDelay, 1 ) ;
                                sendMessage( SubscriptionRqst, Unicast, 0, id, recepient ) ;
                                set( waitDelay, 60 )

                        :: sendRenewal == 1 && waitDelay.val <= 0 ->
                                delay( nodeDelay, 1 ) ;
                                sendMessage( SubscriptionRenewal, Unicast, 0, id, recepient ) ;
                                set( waitDelay, 60 )

                        :: waitUpdate == 1 && updateDelay.val <= 0 ->
                                delay( nodeDelay, 1 ) ;
                                sendMessage( ServiceUpdateRqst, Unicast, 0, id, recepient ) ;
                                waitUpdate = 2          /* NOW WHAT? */
                        od
        fi
}

proctype Node3D( byte id, leader )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        timer nodeDelay ;
        timer updateDelay ;

        set( updateDelay, 30 ) ;
        currentService = 1 ;
        numUpdated = 0 ;

        do
        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                delay( nodeDelay, 1 ) ;

                atomic
                {
                        if
                        :: transType == Broadcast -> skip

                        :: transType == Unicast ->
```

```
                                        if
                                        :: msgType == ServiceUpdateRqst ->
                                                sendMessage( ServiceUpdate, Unicast, currentService, id,
leader ) ;
                                                set( updateDelay, 30 )

                                        :: msgType == HelloDevice ->
                                                sendMessage( HelloCentral, Unicast, 0, id, leader ) ;

                                        :: else
                                        fi
                                fi ;

                                msgType = 0 ;
                                transType = 0 ;
                                msg = 0 ;
                                src = 0 ;
                                dest = 0
                        }

        :: updateDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;

                        if
                        :: numUpdated == 2 ->
                                if
//                              :: currentService == 0 -> currentService = 1 ; numUpdated = 0
                                :: currentService == 1 -> currentService = 0 ; numUpdated = 0
                                :: else
                                fi
                        :: else
                        fi ;

                        sendMessage( ServiceUpdate, Unicast, currentService, id, leader ) ;
                        set( updateDelay, 30 ) ;
        od
}

init
{
        atomic
        {
                run Node300D( 0, 1, 3 ) ;
                run Node300D( 1, 2, 0 ) ;
                run Node300D( 2, 2, 0 ) ;
                run Node3D( 3, 0 )
        }
}




// 2-party Consistency Maintenance with no message loss
// Ceryen Tan, Vasughi Sundramoorthy
```

```
#include "dtime.h"

#define N 4
#define MAX_CH 5

#define Unicast 0
#define Broadcast 1

mtype =
	{
		SrvReg, SrvRegRqst,
		ServiceUpdate, ServiceUpdateRqst,
		HelloDevice, HelloCentral,
		SubscriptionRqst, SubscriptionRqstAck, ServiceDown, SubscriptionRenewal,
Resubscribe,
		SrvSearch, SrvFound, SrvNotFound
	 } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte, byte } ;

byte currentService ;
byte numUpdated ;
bit found300D = 1 ;

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0;
	req = 0
}

inline sendMessage( msgType, transType, aMsg, aMsg1, aSrc, aDest )  /* aMsg represent an attribute, and
aMsg1 is the search requirement */
{
	atomic
	{
		if
		:: transType == Unicast -> from_mail[aDest] ! msgType, Unicast, aMsg, aMsg1, aSrc,
aDest ;

		:: transType == Broadcast ->
			dest = 0 ;

			do
			:: dest < N ->
				if
				:: dest != id -> from_mail[dest] ! msgType, transType, aMsg, aMsg1,
aSrc, dest
				:: else
				fi ;

				dest++
			:: dest >= N -> dest = 0 ; break
			od ;
```

```
                fi
        }
}

proctype Node300D( byte id, mode, recepient )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte req ;
        byte src ;
        byte dest ;

        /* MODE 1 */
        byte waitUpdate ;
        byte waitHello ;
        byte Poll;

        bit subscribe1 ;

        timer renewal1 ;
        timer renewal2 ;

        bit waitAck1 ;
        bit waitAck2 ;

        bit wait300D ;
        timer talk300D ;
        bit reg300D ;
        bit suspendService ;
        byte renew1;
        byte renew;

        timer talk1 ;
        timer talk2 ;

        byte service ;

        /* MODE 2 */
        bit sendSubRqst ;
        bit sendRenewal ;

        timer updateDelay ;

        timer nodeDelay ;
        timer waitDelay ;
    timer searchDelay ;

        if
        :: mode == 1 -> /*Central mode*/
                Poll = 0 ;
```

```
		do
		:: from_mail_to_i ? msgType, transType, msg, req, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: transType == Broadcast -> skip

				:: transType == Unicast ->
					if
					:: msgType == HelloCentral ->
						/* TO DO! Service still available -> Ask to re-
register */
						sendMessage( SrvRegRqst, Unicast, 0, 0, id, 1 ) ;  /*
re-registration for node 1 */
						waitHello = 0 ;

					:: msgType == ServiceDown ->
						Poll = 1;
						suspendService = 0;				/*
deactivate: suspend propagating service information */

					:: msgType == SrvReg ->
						if
						:: src == 2 ->
							printf("MSC: REG 300D\n");
							reg300D = 1 ;
							suspendService = 1; /* activate: service can
be propagated to interested SU */
							if
							:: wait300D == 1 -> sendMessage(
SrvFound, Unicast, 1, 0, id, 1 ) ; wait300D = 0
							:: else
							fi
						:: else
						fi
					:: msgType == SrvSearch ->
						if
						:: req == 0 && reg300D == 1 &&  suspendService  -
> sendMessage( SrvFound, Unicast, 0, 1, id, 1 )
						:: else -> sendMessage( SrvNotFound, Unicast, 0, 0,
id, 1 ) ;

							if
							:: req == 0 -> wait300D = 1
							:: else
							fi
						fi

					:: else
					fi
				fi ;
			}

		/* wait for prompt from SU that node is down */
```

```
                :: Poll == 1 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( HelloDevice, Unicast, 0, 0, id, 2) ;
                        waitHello = 1 ;
                        set( waitDelay, 30 )

                /*poll SM */
                :: waitHello == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( HelloDevice, Unicast, 0, 0, id, 2 ) ;
                        waitHello = 2 ;
                        set( waitDelay, 30 )

                /*poll again SM*/
                :: waitHello == 2 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;                         /* NOW WHAT? -> should purge
data in RL */
                        waitUpdate = 0 ;
                        waitHello = 0 ;
                        printf("MSC: Purged node!") ;
                        reg300D = 0;

                od

        :: mode == 2 -> /* Subscriber SU */
                sendMessage( SubscriptionRqst, Unicast, 0, 0, id, 2 ) ;
                sendSubRqst = 1 ;
                sendRenewal = 0 ;
                waitUpdate = 0 ;
                subscribe1 = 0;
                set( waitDelay, 60 ) ;

                do
                :: from_mail_to_i ? msgType, transType, msg, req, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast -> skip

                                :: transType == Unicast ->
                                        if
                                        :: msgType == SubscriptionRqstAck ->
                                                sendSubRqst = 0 ;
                                                sendRenewal = 1 ;
                                                waitUpdate = 1 ;
                                                set( updateDelay, 30 );
                                                set( waitDelay, 20 )

                                        :: msgType == ServiceUpdate ->
                                                printf("MSC: %d\n", msg ) ;

                                                if
                                                :: service != msg && msg == currentService ->
numUpdated++
```

```
                                                :: else
                                                fi ;
                                                subscribe1 = 1;
                                                service = msg ;
                                                waitUpdate = 1 ;
                                                set( updateDelay, 30 )

                                        :: msgType == Resubscribe ->
                                                sendMessage( SubscriptionRqst, Unicast, 0, 0, id, 2 )
;
                                                sendSubRqst = 1 ;
                                                sendRenewal = 0 ;
                                                waitUpdate = 0 ;
                                                reset( updateDelay ) ;
                                                set( waitDelay, 20 )

                                        :: msgType == SrvFound ->
                                                printf("MSC: FOUND\n") ;
                                                found300D = 1;

                                                if          /* check if the msg in SrvFound provides the
correct req and update the service attr */
                                                :: req == 0 && service != msg -> numUpdated++
        /* for verification -> []<>(numUpdated >=1)*/
                                                :: else -> wait300D = 1
                                                fi

                                        :: msgType == SrvNotFound ->
                                                printf("MSC: NOT FOUND\n") ;
                                                wait300D = 1;

                                                :: else
                                                fi
                                fi ;

                                clearMail()
                        }

                :: sendSubRqst == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( SubscriptionRqst, Unicast, 0, 0, id, 2) ;
                        set( waitDelay, 60 )

                :: sendRenewal == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( SubscriptionRenewal, Unicast, 0, 0, id, 2 ) ;
                        set( waitDelay, 20 )

                :: waitUpdate == 1 && updateDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( ServiceUpdateRqst, Unicast, 0, 0, id, 2 ) ;
                        waitUpdate = 2;
                        set(updateDelay, 30)                    /* NOW WHAT? -> Send msg to Central
notifying service down */

                :: waitUpdate == 2 && updateDelay.val <= 0 ->
```

```
                        delay( nodeDelay, 1 ) ;
                        subscribe1=0;
                        sendMessage( ServiceDown, Unicast, 0, 0, id, 0);
                        found300D = 0 ;
                        set( searchDelay, 30 )

                :: waitUpdate == 2 && subscribe1 == 1 -> /* SU rx ServiceUpdate after it thinks SM is
unavailable */
                        delay (nodeDelay, 1);
                        waitUpdate = 1;
                        sendMessage( SubscriptionRenewal, Unicast, 0, 0, id, 2 );
                        waitUpdate = 1;
                        set(updateDelay, 30)

                :: found300D==0 && searchDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( SrvSearch, Unicast, 0, 0, id, 0 ) ;
                        found300D = 0 ;
                        set( searchDelay, 30 )
                od

        :: mode == 3 -> /* Service Provider SM */
                set( updateDelay, 30 ) ;
                currentService = 1 ;
                numUpdated = 0 ;

                do
                :: from_mail_to_i ? msgType, transType, msg, req, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast -> skip

                                :: transType == Unicast ->
                                        if

                                        :: msgType == SubscriptionRqst ->
                                                subscribe1 = 1 ;
                                                renew=1;
                                                sendMessage( SubscriptionRqstAck, Unicast, 0, 0, id,
1 ) ;
                                                sendMessage( ServiceUpdate, Unicast,
currentService, 0, id, 1 );
                                                waitUpdate = 1 ;
                                                set( renewal1, 40 )

                                        :: msgType == SubscriptionRenewal ->
                                                if
                                                :: subscribe1 == 1 -> set( renewal1, 40 )
                                                :: else -> sendMessage( Resubscribe, Unicast, 0, 0,
id, 1 )
                                                fi

                                        :: msgType == ServiceUpdateRqst ->
```

```
                                                sendMessage( ServiceUpdate, Unicast,
currentService, 0, id, 2) ;
                                                set( updateDelay, 30 )

                                        :: msgType == HelloDevice ->
                                                sendMessage( HelloCentral, Unicast, 0, 0, id, 0) ;

                                        :: msgType == SrvRegRqst -> /* Re-registration */
                                                sendMessage( SrvReg, Unicast, currentService, 0, id,
0 )

                                        :: else

                                        fi
                                fi ;

                                clearMail()
                        }

                :: subscribe1==1 && updateDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;

                        if
                        :: numUpdated == 1 ->
                                if
                                :: currentService == 1 -> currentService = 0;
                                :: else
                                fi
                        :: else
                        fi ;

                        sendMessage( ServiceUpdate, Unicast, currentService, 0, id, 1 ) ;
                        set( updateDelay, 40 )

                :: renew==1 && renewal1.val <= 0  ->
                        delay( nodeDelay, 1);
                        renew=2;
                        printf("MSC: RENEW1 NOT RECEIVED\n") ;
                        set (renewal1, 40)

                :: renew==2 && renewal1.val <=0 ->
                        delay( nodeDelay, 1);
                        renew=0;
                        printf("MSC: RENEW2 NOT RECEIVED\n") ;
                        subscribe1=0;
                od
        fi
}

init
{
        atomic
        {
                run Node300D( 0, 1, 0 ) ;
                run Node300D( 1, 2, 0 ) ;
                run Node300D( 2, 3, 0 ) ;
```

```
        }
}
```

```
// Multiple Leader Election (Central Negotiation) with no message loss
// Ceryen Tan, Vasughi Sundramoorthy
// 8/20 - Clear buffer error
// 8/17 - Cleanup attempt, Message renaming

#include "dtime.h"

#define N 4
#define MAX_CH 6

#define Unicast 0
#define Broadcast 1

mtype =
        {
                IAmCentral, CentralNego, LeaderElect,
                NewCentralAssign, CentralAssignAck,
                BackupAssign, BackupAssignAck, HelloCentral, HelloDevice, BackupCancel,
RankEntryAdd, MoreCentrals,
                MyResRqst, MyResReply,
                SrvSearch, SrvSearchReply
        } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
chan nego_list[N] = [N] of {byte, bit, bit } ;

byte nr_leader = 0 ;
byte nr_backup = 0 ;
byte rightAssigned = 0 ;

bit leader[N] ;

typedef rankEntry
```

```
{
	byte id, rank ;
} ;

inline addRankEntry( anID )
{
	if
	:: anID != id ->
		if
		:: anID == 0 -> updateRanklist( 10, 0 )
		:: anID == 1 -> updateRanklist( 4, 1 )
		:: anID == 2 -> updateRanklist( 8, 2 )
		:: anID == 3 -> updateRanklist( 6, 3 )
		fi
	:: else
	fi
}

inline checkRanklist( anID )
{
	atomic
	{
		findInRanklist( anID ) ;

		if
		:: tempLoc == 255 -> sendMessage( MyResRqst, Unicast, 0, id, anID )
		:: else -> tempLoc = 0
		fi
	}
}

inline clearBuffer()
{
	atomic
	{
		do
		:: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
		:: empty(from_mail_to_i) -> break
		od
	}
}

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0
}

inline clearRanklist()
{
	atomic
	{
		counter = 0 ;
```

```
                do
                :: counter < N -> rankList[counter].id = 255 ; rankList[counter].rank = 255 ; counter++
                :: counter == N -> counter = 0 ; break
                od
        }
}

inline clearVariables()
{
        announceReceived = 0 ;
        negoReceived = 0 ;

        mode = 0 ;
        centralNego = 0 ;

        assignBackup = 0 ;
        backupAssigned = 0 ;
        waitBackupReply = 0 ;
        backupRank = 0 ;
        ranklistEmpty = 0 ;

        waitAppointAck = 0 ;
        lastAppointed = 255 ;

        lastCentral = 255 ;
        lastBackup = 255 ;
        sendHelloBackup = 0 ;
        sendHelloCentral = 0 ;

        reset( waitDelay ) ;
        reset( nodeDelay ) ;
        reset( backupDelay ) ;
        reset( centralNegoMax ) ;
        reset( centralAppointDelay ) ;

        do
        :: nempty( my_nego_list ) -> my_nego_list ? _, _, _
        :: empty( my_nego_list ) -> break
        od
}

inline findHighest()
{
        atomic
        {
                counter = 0 ;
                aRank = 0 ;
                anID = 255 ;

                do
                :: counter < N ->
                        if
                        :: rankList[counter].rank != 255 && rankList[counter].rank > aRank ->
                                aRank = rankList[counter].rank ;
                                anID = rankList[counter].id
                        :: else
```

```
                        fi ;
                        counter++
                :: counter == N -> counter = 0 ; break
                od
        }
}

inline findInRanklist( anID )
{
        atomic
        {
                tempLoc = 255 ;
                counter = 0 ;
                do
                :: counter < N ->
                        if
                        :: rankList[counter].id == anID -> tempLoc = counter ; break
                        :: else
                        fi ;
                        counter++
                :: counter == N -> counter = 0 ; break
                od

        }
}

inline pruneRanklist( aRank, anID )
{
        atomic
        {
                counter = 0 ;

                do
                :: counter < N ->
                        if
                        :: rankList[counter].rank > aRank || rankList[counter].id == anID ->
                                rankList[counter].rank = 255 ;
                                rankList[counter].id = 255
                        :: else
                        fi ;
                        counter++
                :: counter >= N -> counter = 0 ; break
                od
        }
}

inline removeEntry( anID )
{
        atomic
        {
                counter = 0 ;

                do
                :: counter < N ->
                        if
```

```
                        :: rankList[counter].id == anID -> rankList[counter].id = 255;
rankList[counter].rank = 255
                        :: else
                        fi ;
                        counter++
                :: counter == N -> counter = 0 ; break
                od
        }
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
        atomic
        {
                if
                :: transType == Unicast -> from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                :: transType == Broadcast ->
                        dest = 0 ;

                        do
                        :: dest < N ->
                                if
                                :: dest != id -> from_mail[dest] ! msgType, transType, aMsg, aSrc, dest
;
                                :: else
                                fi ;

                                dest++
                        :: dest >= N -> dest = 0 ; break
                        od ;
                fi
        }
}

inline sendRankEntry( toSend, dest )
{
        if
        :: lastBackup != 255 && backupAssigned == 1 -> sendMessage( RankEntryAdd, Unicast, toSend,
id, dest )
        :: else
        fi
}

inline sendRanklist( dest )
{
        if
        :: lastBackup != 255 && backupAssigned == 1 ->
                atomic
                {
                        counter = 0 ;

                        do
                        :: counter < N ->
                                if
                                :: rankList[counter].id != 255 -> sendMessage( RankEntryAdd,
Unicast, counter, id, dest )
```

```
                                        :: else
                                        fi ;
                                        counter++

                                :: counter >= N -> counter = 0 ; break
                                od
                        }
                :: else
                fi
}

inline updateRanklist( aRank, anID )
{
        atomic
        {
                counter = 0 ;
                tempLoc = 255 ;

                do
                :: counter < N ->
                        if
                        :: rankList[counter].id == anID -> tempLoc = counter ; break
                        :: else ->
                                if
                                :: tempLoc == 255 &&
                                        rankList[counter].rank == 255 &&
                                        rankList[counter].id == 255 ->
                                        tempLoc = counter
                                :: else -> counter++
                                fi
                        fi ;
                :: counter == N -> break
                od ;

                if
                :: tempLoc == 255 -> printf("MSC: Full Ranklist\n")
                :: else ->
                        rankList[tempLoc].rank = aRank ;
                        rankList[tempLoc].id = anID ;
                fi ;

                counter = 0 ;
                tempLoc = 255
        }
}

proctype Node( byte id, rank )
{
        /* Mail */
        chan from_mail_to_i = from_mail[id] ;
        /*xr from_mail_to_i ;*/

        /* Messages */
        mtype msgType ;
        byte transType ;
        byte msg ;
```

```
	byte src ;
	byte dest ;

	/* Central negotiation table */
	chan my_nego_list = nego_list[id] ;
	xr my_nego_list ;
	xs my_nego_list ;

	bit announceReceived ;
	bit negoReceived ;

	/* Central flags */
	byte mode ;
	bit centralNego ;
	bit startup ;

	bit assignBackup ;
	bit backupAssigned ;
	byte waitBackupReply ;
	byte backupRank ;
	bit ranklistEmpty ;

	/* Central appoint */
	byte waitAppointAck ;
	byte lastAppointed ;

	/* Normal flags */
	byte lastCentral ;

	/* Backup flags */
	byte lastBackup ;
	byte sendHelloBackup ;
	bit sendHelloCentral ;

	/* Timers */
	timer waitDelay ;
	timer nodeDelay ;
	timer backupDelay ;
	timer centralNegoMax ;
	timer centralAppointDelay ;

	/* Ranklist */
	rankEntry rankList[N] ;
	byte counter ;
	byte tempLoc ;
	byte aRank, anID ;

	printf("MSC: START\n") ;
	atomic { clearBuffer() } ;

start:
	atomic
	{
		clearRanklist() ;
		clearVariables() ;
```

```
		startup = 1 ;

		if
		:: leader[id] == 1 ->
			mode = 2 ;

			if
			:: skip -> updateRanklist( 10, 0 )
			:: false
			fi ;

			if
			:: skip -> updateRanklist( 4, 1 )
			:: false
			fi ;

			if
			:: skip -> updateRanklist( 8, 2 )
			:: false
			fi ;

			if
			:: skip -> updateRanklist( 6, 3 )
			:: false
			fi ;

			pruneRanklist( rank, id )
		:: else -> mode = 3
		fi
} ;

do
:: mode == 2 ->
		printf("MSC: LEADER\n") ;

		if
		:: id == 0 -> rightAssigned = 1
		:: else
		fi ;

		startup = 0 ;
		assignBackup = 1 ;
		set( waitDelay, 30 ) ;

		nr_leader++ ;

		do
		:: from_mail_to_i ? msgType, transType, msg, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: transType == Broadcast ->
					if
					:: src != id ->
```

```
                                        if
                                        :: msgType == IAmCentral ->
                                                announceReceived = 0 ;
                                                negoReceived = 0 ;

                                                if
                                                :: my_nego_list ?? [eval(src),
announceReceived, negoReceived] ->
                                                        my_nego_list ?? src,
announceReceived, negoReceived
                                                :: else
                                                fi ;

                                                if
                                                :: !(announceReceived == 1 &&
negoReceived == 1) ->
                                                        sendMessage( CentralNego,
Unicast, rank, id, src ) ;
                                                        centralNego = 1 ;
                                                        reset( waitDelay ) ;
                                                        set( centralNegoMax, 30 ) ;
                                                :: else
                                                fi ;

                                                announceReceived = 1 ;
                                                negoReceived = 0 ;
                                                my_nego_list ! src, announceReceived,
negoReceived ;

                                                announceReceived = 0 ;
                                                negoReceived = 0

                                        :: msgType == LeaderElect ->
                                                sendMessage( IAmCentral, Unicast, 0, id,
255 ) ;
                                                checkRanklist( src )

                                        :: else -> checkRanklist( src )
                                        fi
                                :: else
                                fi

                        :: transType == Unicast ->
                                if
                                :: msgType == BackupAssignAck ->
                                        if
                                        :: src == lastBackup ->
                                                assignBackup = 0 ;
                                                backupAssigned = 1 ;
                                                waitBackupReply = 0 ;
                                                sendHelloBackup = 1 ;
                                                set( backupDelay, 15 ) ;

                                                sendRanklist( lastBackup )
                                        :: else ->
```

```
                                        sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
                                        checkRanklist( src )
                                fi

                        :: msgType == CentralAssignAck ->
                                if
                                :: src == lastAppointed ->
                                        if
                                        :: backupAssigned == 1 ->
                                                sendMessage( BackupCancel,
Unicast, 0, id, lastBackup ) ;
                                        :: else
                                        fi ;

                                        sendRanklist( lastAppointed ) ;

                                        clearVariables() ;
                                        lastCentral = src ;

                                        if
                                        :: id == 0 -> rightAssigned = 0
                                        :: else
                                        fi ;

                                        mode = 4 ;
                                        nr_leader-- ;
                                        leader[id] = 0 ;
                                        break
                                :: else
                                fi

                        :: msgType == CentralNego ->
                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                                        my_nego_list ?? src, announceReceived,
negoReceived
                                :: else
                                fi ;

                                if
                                :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                        sendMessage( CentralNego, Unicast, rank,
id, src ) ;
                                :: else
                                fi ;

                                negoReceived = 1 ;
                                my_nego_list ! src, announceReceived,
negoReceived ;
```

```
                                        announceReceived = 0 ;
                                        negoReceived = 0 ;

                                        if
                                        :: msg > rank ->
                                                printf( "MSC: LOST\n" ) ;
                                                if
                                                :: backupAssigned == 1 ->
                                                        sendMessage( BackupCancel,
Unicast, 0, id, lastBackup )
                                                :: else
                                                fi ;

                                                clearVariables() ;
                                                clearRanklist() ;

                                                if
                                                :: id == 0 -> rightAssigned = 0
                                                :: else
                                                fi ;

                                                mode = 3 ;
                                                nr_leader-- ;
                                                leader[id] = 0 ;
                                                break
                                        :: else ->
                                                if
                                                :: centralNego == 0 ->
                                                        centralNego = 1 ;
                                                        reset( waitDelay ) ;
                                                        set( centralNegoMax, 30 )
                                                :: else
                                                fi
                                        fi ;

                                        if
                                        :: backupAssigned == 1 && msg > backupRank ->
                                                sendMessage( BackupCancel, Unicast, 0, id,
lastBackup ) ;
                                                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;

                                                lastBackup = src ;
                                                backupRank = msg ;
                                                assignBackup = 0 ;
                                                backupAssigned = 0 ;
                                                waitBackupReply = 1 ;
                                                sendHelloBackup = 0 ;
                                                set( backupDelay, 15 )
                                        :: else
                                        fi ;

                                        if
                                        :: ranklistEmpty == 1 ->
                                                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;
```

```
                                        lastBackup = src ;
                                        backupRank = msg ;
                                        assignBackup = 0 ;
                                        waitBackupReply = 1 ;
                                        sendHelloBackup = 0 ;
                                        set( backupDelay, 30 ) ;
                                        ranklistEmpty = 0
                                :: else
                                fi ;

                                updateRanklist( msg, src ) ;
                                sendRankEntry( src, lastBackup )

                        :: msgType == HelloCentral ->
                                if
                                :: src == lastBackup ->
                                        sendHelloBackup = 1 ;
                                        set( backupDelay, 15 )
                                :: else ->
                                        sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
                                        checkRanklist( src )
                                fi

                        :: msgType == MoreCentrals ->
                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                                        my_nego_list ?? src, announceReceived,
negoReceived
                                :: else
                                fi ;

                                if
                                :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                        sendMessage( CentralNego, Unicast, rank,
id, msg ) ;
                                :: else
                                fi ;

                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: src != lastBackup ->
                                        sendMessage( BackupCancel, Unicast, rank,
id, src )
                                :: else
                                fi ;

                                checkRanklist( src )
```

```
                    :: msgType == MyResReply ->
                            if
                            :: msg > rank ->
                                    sendMessage( NewCentralAssign, Unicast,
rank, id, src ) ;

                                    lastAppointed = src ;
                                    waitAppointAck = 1 ;
                                    set( centralAppointDelay, 30 )

                            :: else
                            fi ;

                            if
                            :: backupAssigned == 1 && msg > backupRank ->
                                    sendMessage( BackupCancel, Unicast, 0, id,
lastBackup ) ;
                                    sendMessage( BackupAssign, Unicast, 0, id,
src ) ;

                                    lastBackup = src ;
                                    backupRank = msg ;
                                    assignBackup = 0 ;
                                    backupAssigned = 0 ;
                                    waitBackupReply = 1 ;
                                    sendHelloBackup = 0 ;
                                    set( backupDelay, 15 )
                            :: else
                            fi ;

                            if
                            :: ranklistEmpty == 1 ->
                                    sendMessage( BackupAssign, Unicast, 0, id,
src ) ;

                                    lastBackup = src ;
                                    backupRank = msg ;
                                    assignBackup = 0 ;
                                    backupAssigned = 0 ;
                                    waitBackupReply = 1 ;
                                    sendHelloBackup = 0 ;
                                    set( backupDelay, 30 ) ;
                                    ranklistEmpty = 0
                            :: else
                            fi ;

                            updateRanklist( msg, src ) ;
                            sendRankEntry( src, lastBackup )

                    :: msgType == RankEntryAdd ->
                            if
                            :: src == lastBackup ->
                                    addRankEntry( msg )
                            :: else
                            fi
```

```
                                        :: msgType == SrvSearch ->
                                                /*sendMessage( SrvSearchReply, Unicast, 0, id, src )
;*/
                                                checkRanklist( src )

                                        :: else -> checkRanklist( src )
                                        fi
                                fi ;

                                clearMail()
                        }

                /*** Announcements ***/
                :: centralNego == 0 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;
                        set( waitDelay, 30 )

                /*** Backup assignment ***/
                :: centralNego == 0 && assignBackup == 1 ->
                        delay( nodeDelay, 1 ) ;
                        findHighest() ;

                        if
                        :: anID == 255 ->
                                printf( "MSC: Empty ranklist\n" ) ;
                                ranklistEmpty = 1 ;
                                assignBackup = 0 ;
                                backupAssigned = 0 ;
                                waitBackupReply = 0 ;
                                lastBackup = 255 ;
                                backupRank = 0 ;
                                sendHelloBackup = 0
                        :: else ->
                                lastBackup = anID ;
                                backupRank = aRank ;

                                sendMessage( BackupAssign, Unicast, 0, id, anID ) ;
                                assignBackup = 0 ;
                                backupAssigned = 0 ;
                                waitBackupReply = 1 ;
                                sendHelloBackup = 0 ;
                                set( backupDelay, 30 ) ;
                        fi

                :: waitBackupReply == 1 && backupDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( BackupAssign, Unicast, 0, id, lastBackup ) ;
                        assignBackup = 0 ;
                        backupAssigned = 0 ;
                        waitBackupReply = 2 ;
                        sendHelloBackup = 0 ;
                        set( backupDelay, 30 ) ;

                :: waitBackupReply == 2 && backupDelay.val <= 0 ->
```

```
            delay( nodeDelay, 1 ) ;
            printf("MSC: Remove\n") ;
            removeEntry( lastBackup ) ;
            lastBackup = 255 ;
            backupRank = 0 ;
            assignBackup = 1 ;
            backupAssigned = 0 ;
            waitBackupReply = 0 ;
            sendHelloBackup = 0 ;

    /*** Backup polling ***/
    :: sendHelloBackup == 1 && backupDelay.val <= 0 ->
            delay( nodeDelay, 1 ) ;
            sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
            sendHelloBackup = 2 ;
            set( backupDelay, 30 )
    :: sendHelloBackup == 2 && backupDelay.val <= 0 ->
            delay( nodeDelay, 1 ) ;
            sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
            sendHelloBackup = 3 ;
            set( backupDelay, 30 )
    :: sendHelloBackup == 3 && backupDelay.val <= 0 ->
            delay( nodeDelay, 1 ) ;
            removeEntry( lastBackup ) ;
            lastBackup = 255 ;
            backupRank = 0 ;
            assignBackup = 1 ;
            backupAssigned = 0 ;
            waitBackupReply = 0 ;
            sendHelloBackup = 0 ;

    /*** Central negotiation max timeout ***/
    :: centralNego == 1 && centralNegoMax.val <= 0 ->
            if
            :: from_mail_to_i ?? [eval(IAmCentral), transType, msg, src, dest] ||
                    from_mail_to_i ?? [eval(CentralNego), transType, msg, src, dest] ->
                        set( centralNegoMax, 30 )
            :: else ->
                    centralNego = 0 ;
                    reset( waitDelay ) ;

                    do
                    :: my_nego_list ? _, _, _
                    :: empty( my_nego_list ) -> break
                    od
            fi

    /*** Central appointment ***/
    :: waitAppointAck == 1 && centralAppointDelay.val <= 0 ->
            sendMessage( NewCentralAssign, Unicast, rank, id, lastAppointed ) ;
            waitAppointAck = 2 ;
            set( centralAppointDelay, 30 ) ;
    :: waitAppointAck == 2 && centralAppointDelay.val <= 0 ->
            waitAppointAck = 0 ;
            lastAppointed = 255
    od ;
```

```
:: mode == 3 ->
        printf("MSC: NORMAL\n") ;

        if
        :: startup == 1 ->
                if
                :: leader[0] == 1 -> lastCentral = 0
                :: leader[1] == 1 -> lastCentral = 1
                :: leader[2] == 1 -> lastCentral = 2
                :: leader[3] == 1 -> lastCentral = 3
                :: else -> printf("MSC: ERROR\n")
                fi ;

                set( waitDelay, 30 ) ;
                startup = 0
        :: else -> lastCentral = 255
        fi ;

        do
        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                delay( nodeDelay, 1 ) ;

                atomic
                {
                        if
                        :: msgType == IAmCentral ->
                                lastCentral = src ;

                        :: msgType == BackupAssign ->
                                if
                                :: lastCentral != 255 && src != lastCentral ->
                                        sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
                                        sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                :: else ->
                                        sendMessage( BackupAssignAck, Unicast, 0, id, src )
;

                                        clearVariables() ;
                                        lastCentral = src ;

                                        mode = 4 ;
                                        break
                                fi

                        :: msgType == NewCentralAssign ->
                                sendMessage( CentralAssignAck, Unicast, 0, id, src ) ;

                                clearVariables() ;
                                lastBackup = src ;
                                backupRank = msg ;
                                sendHelloBackup = 1 ;
                                assignBackup = 0 ;
                                backupAssigned = 1 ;
```

```
                                        mode = 2 ;
                                        leader[id] = 1 ;
                                        break

                                :: msgType == MyResRqst ->
                                        sendMessage( MyResReply, Unicast, rank, id, src ) ;

                                :: else
                                fi ;

                                clearMail()
                        }

                :: lastCentral != 255 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( SrvSearch, Unicast, 0, id, lastCentral ) ;
                        set( waitDelay, 30 )
                od

        :: mode == 4 ->
                printf("MSC: BACKUP\n") ;
                nr_backup++ ;
                set( waitDelay, 60 ) ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
//                              :: msgType == IAmCentral ->
//                                      if
//                                      :: src != lastCentral ->
//                                              sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
//                                              sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
//                                      :: else
//                                      fi

                                :: msgType == BackupAssign ->
                                        if
                                        :: src == lastCentral ->
                                                sendMessage( BackupAssignAck, Unicast, 0, id, src )
                                        :: else ->
                                                sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
                                                sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                        fi

                                :: msgType == BackupCancel ->
                                        clearRanklist() ;
                                        clearVariables() ;
```

```
                                        mode = 3 ;
                                        nr_backup-- ;
                                        break

                                :: msgType == HelloDevice ->
                                        if
                                        :: src == lastCentral ->
                                                sendMessage( HelloCentral, Unicast, 0, id,
lastCentral ) ;

                                                sendHelloCentral = 0 ;
                                                set( waitDelay, 60 )
                                        :: else //->
//                                              sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
//                                              sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                        fi

                                :: msgType == RankEntryAdd ->
                                        if
                                        :: src == lastCentral ->
                                                addRankEntry( msg )
                                        :: else //->
//                                              sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
//                                              sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                        fi

                                :: else
                                        fi ;

                                        clearMail()
                        }

                /*** Central polling ***/
                :: sendHelloCentral == 0 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( HelloCentral, Unicast, 0, id, lastCentral ) ;
                        sendHelloCentral = 1 ;
                        set( waitDelay, 60 )
                :: sendHelloCentral == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;

                        clearVariables() ;

                        mode = 2 ;
                        nr_backup-- ;
                        leader[id] = 1 ;
                        break
                od
        od
}
```

```
init
{
	atomic
	{
		if
		:: leader[0] = 1 ; nr_leader++
		:: false
		fi ;

		if
		:: leader[1] = 1 ; nr_leader++
		:: false
		fi ;

		if
		:: leader[2] = 1 ; nr_leader++
		:: false
		fi ;

		if
		:: leader[3] = 1 ; nr_leader++
		:: false
		fi ;

		if
		:: nr_leader == 0 ->
			if
			:: leader[0] = 1
			:: leader[1] = 1
			:: leader[2] = 1
			:: leader[3] = 1
			fi
		:: else
		fi ;

		nr_leader = 0 ;

		run Node(0, 10) ;
		run Node(1, 4) ;
		run Node(2, 8) ;
		run Node(3, 6)
	}
}
```

```
// Multiple Leader Election (Central Negotiation) with message loss
// Ceryen Tan, Vasughi Sundramoorthy
```

```
// 8/20 - Clear buffer error
// 8/17 - Cleanup attempt, Message renaming

#include "dtime.h"

#define N 4
#define MAX_CH 6
#define MAX_LOSS 1

#define Unicast 0
#define Broadcast 1

mtype =
	{
		IAmCentral, CentralNego, LeaderElect,
		NewCentralAssign, CentralAssignAck,
		BackupAssign, BackupAssignAck, HelloCentral, HelloDevice, BackupCancel,
RankEntryAdd, MoreCentrals,
		MyResRqst, MyResReply,
		SrvSearch, SrvSearchReply
	} ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
chan nego_list[N] = [N] of {byte, bit, bit } ;

byte nr_leader = 0 ;
byte nr_backup = 0 ;
byte rightAssigned = 0 ;

byte lossCounters[15] ;
bit leader[N] ;

typedef rankEntry
{
	byte id, rank ;
} ;

inline addRankEntry( anID )
{
	if
	:: anID != id ->
		if
		:: anID == 0 -> updateRanklist( 10, 0 )
		:: anID == 1 -> updateRanklist( 4, 1 )
		:: anID == 2 -> updateRanklist( 8, 2 )
		:: anID == 3 -> updateRanklist( 6, 3 )
		fi
	:: else
	fi
}

inline checkRanklist( anID )
{
	atomic
	{
		findInRanklist( anID ) ;
```

```
                    if
                    :: tempLoc == 255 -> sendMessage( MyResRqst, Unicast, 0, id, anID )
                    :: else -> tempLoc = 0
                    fi
            }
}

inline clearBuffer()
{
            atomic
            {
                    do
                    :: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
                    :: empty(from_mail_to_i) -> break
                    od
            }
}

inline clearMail()
{
            msgType = 0 ;
            transType = 0 ;
            msg = 0 ;
            src = 0 ;
            dest = 0 ;
            type = 0
}

inline clearRanklist()
{
            atomic
            {
                    counter = 0 ;
                    do
                    :: counter < N -> rankList[counter].id = 255 ; rankList[counter].rank = 255 ; counter++
                    :: counter == N -> counter = 0 ; break
                    od
            }
}

inline clearVariables()
{
            announceReceived = 0 ;
            negoReceived = 0 ;

            mode = 0 ;
            centralNego = 0 ;

            assignBackup = 0 ;
            backupAssigned = 0 ;
            waitBackupReply = 0 ;
            backupRank = 0 ;
            ranklistEmpty = 0 ;

            waitAppointAck = 0 ;
            lastAppointed = 255 ;
```

```
	lastCentral = 255 ;
	lastBackup = 255 ;
	sendHelloBackup = 0 ;
	sendHelloCentral = 0 ;

	reset( waitDelay ) ;
	reset( nodeDelay ) ;
	reset( backupDelay ) ;
	reset( centralNegoMax ) ;
	reset( centralAppointDelay ) ;

	do
	:: nempty( my_nego_list ) -> my_nego_list ? _, _, _
	:: empty( my_nego_list ) -> break
	od
}

inline findHighest()
{
	atomic
	{
		counter = 0 ;
		aRank = 0 ;
		anID = 255 ;

		do
		:: counter < N ->
			if
			:: rankList[counter].rank != 255 && rankList[counter].rank > aRank ->
				aRank = rankList[counter].rank ;
				anID = rankList[counter].id
			:: else
			fi ;
			counter++
		:: counter == N -> counter = 0 ; break
		od
	}
}

inline findInRanklist( anID )
{
	atomic
	{
		tempLoc = 255 ;
		counter = 0 ;
		do
		:: counter < N ->
			if
			:: rankList[counter].id == anID -> tempLoc = counter ; break
			:: else
			fi ;
			counter++
		:: counter == N -> counter = 0 ; break
		od
	}
```

```
}

inline getType( msgType )
{
	if
	:: msgType == IAmCentral -> type = 0
	:: msgType == CentralNego -> type = 1
	:: msgType == NewCentralAssign -> type = 2
	:: msgType == BackupAssign -> type = 3
	:: msgType == BackupAssignAck -> type = 4
	:: msgType == HelloCentral -> type = 5
	:: msgType == HelloDevice -> type = 6
	:: msgType == BackupCancel -> type = 7
	:: msgType == MyResRqst -> type = 8
	:: msgType == MyResReply -> type = 9
	:: msgType == SrvSearch -> type = 10
	:: msgType == SrvSearchReply -> type = 11
	:: msgType == LeaderElect -> type = 12
	:: msgType == MoreCentrals -> type = 13
	:: msgType == CentralAssignAck -> type = 14
	:: else -> type = 255
	fi
}

inline pruneRanklist( aRank, anID )
{
	atomic
	{
		counter = 0 ;

		do
		:: counter < N ->
			if
			:: rankList[counter].rank > aRank || rankList[counter].id == anID ->
				rankList[counter].rank = 255 ;
				rankList[counter].id = 255
			:: else
			fi ;
			counter++
		:: counter >= N -> counter = 0 ; break
		od
	}
}

inline removeEntry( anID )
{
	atomic
	{
		counter = 0 ;

		do
		:: counter < N ->
			if
			:: rankList[counter].id == anID -> rankList[counter].id = 255;
rankList[counter].rank = 255
			:: else
```

```
                        fi ;
                        counter++
                :: counter == N -> counter = 0 ; break
                od
        }
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
        getType( msgType ) ;

        atomic
        {
                if
                :: transType == Unicast ->
                        if
                        :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                        :: else ->
                                if
                                :: from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                                :: true -> printf( "MSC: UNICAST: %d, %d, %d\n", aMsg, aSrc, dest )
; lossCounters[type]++
                                fi
                        fi ;
                :: transType == Broadcast ->
                        dest = 0 ;

                        do
                        :: dest < N ->
                                if
                                :: dest != id ->
                                        if
                                        :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                                from_mail[dest] ! msgType, transType, aMsg, aSrc,
dest
                                        :: else ->
                                                if
                                                :: from_mail[dest] ! msgType, transType, aMsg,
aSrc, dest
                                                :: true -> printf( "MSC: BROADCAST: %d, %d,
%d\n", aMsg, aSrc, dest ) ;
                                                        lossCounters[type]++
                                                fi
                                        fi ;
                                :: else
                                fi ;

                                dest++
                        :: dest >= N -> dest = 0 ; break
                        od ;
                fi ;

                type = 0
        }
}
```

```
inline sendRankEntry( toSend, dest )
{
	if
	:: lastBackup != 255 && backupAssigned == 1 -> sendMessage( RankEntryAdd, Unicast, toSend,
id, dest )
	:: else
	fi
}

inline sendRanklist( dest )
{
	if
	:: lastBackup != 255 && backupAssigned == 1 ->
		atomic
		{
			counter = 0 ;

			do
			:: counter < N ->
				if
				:: rankList[counter].id != 255 -> sendMessage( RankEntryAdd,
Unicast, counter, id, dest )
				:: else
				fi ;
				counter++

			:: counter >= N -> counter = 0 ; break
			od
		}
	:: else
	fi
}

inline updateRanklist( aRank, anID )
{
	atomic
	{
		counter = 0 ;
		tempLoc = 255 ;

		do
		:: counter < N ->
			if
			:: rankList[counter].id == anID -> tempLoc = counter ; break
			:: else ->
				if
				:: tempLoc == 255 &&
					rankList[counter].rank == 255 &&
					rankList[counter].id == 255 ->
					tempLoc = counter
				:: else -> counter++
				fi
			fi ;
		:: counter == N -> break
		od ;
```

```
                    if
                    :: tempLoc == 255 -> printf("MSC: Full Ranklist\n")
                    :: else ->
                            rankList[tempLoc].rank = aRank ;
                            rankList[tempLoc].id = anID ;
                    fi ;

                    counter = 0 ;
                    tempLoc = 255
            }
}

proctype Node( byte id, rank )
{
            /* Mail */
            chan from_mail_to_i = from_mail[id] ;
            /*xr from_mail_to_i ;*/

            /* Messages */
            mtype msgType ;
            byte transType ;
            byte msg ;
            byte src ;
            byte dest ;
            byte type ;

            /* Central negotiation table */
            chan my_nego_list = nego_list[id] ;
            xr my_nego_list ;
            xs my_nego_list ;

            bit announceReceived ;
            bit negoReceived ;

            /* Central flags */
            byte mode ;
            bit centralNego ;
            bit startup ;

            bit assignBackup ;
            bit backupAssigned ;
            byte waitBackupReply ;
            byte backupRank ;
            bit ranklistEmpty ;

            /* Central appoint */
            byte waitAppointAck ;
            byte lastAppointed ;

            /* Normal flags */
            byte lastCentral ;

            /* Backup flags */
            byte lastBackup ;
            byte sendHelloBackup ;
```

```
	bit sendHelloCentral ;

	/* Timers */
	timer waitDelay ;
	timer nodeDelay ;
	timer backupDelay ;
	timer centralNegoMax ;
	timer centralAppointDelay ;

	/* Ranklist */
	rankEntry rankList[N] ;
	byte counter ;
	byte tempLoc ;
	byte aRank, anID ;

	printf("MSC: START\n") ;
	atomic { clearBuffer() } ;

start:
	atomic
	{
		clearRanklist() ;
		clearVariables() ;

		startup = 1 ;

		if
		:: leader[id] == 1 ->
			mode = 2 ;

			if
			:: skip -> updateRanklist( 10, 0 )
			:: false
			fi ;

			if
			:: skip -> updateRanklist( 4, 1 )
			:: false
			fi ;

			if
			:: skip -> updateRanklist( 8, 2 )
			:: false
			fi ;

			if
			:: skip -> updateRanklist( 6, 3 )
			:: false
			fi ;

			pruneRanklist( rank, id )
		:: else -> mode = 3
		fi
	} ;

	do
```

```
:: mode == 2 ->
        printf("MSC: LEADER\n") ;

        if
        :: id == 0 -> rightAssigned = 1
        :: else
        fi ;

        startup = 0 ;
        assignBackup = 1 ;
        set( waitDelay, 30 ) ;

        nr_leader++ ;

        do
        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                delay( nodeDelay, 1 ) ;

                atomic
                {
                        if
                        :: transType == Broadcast ->
                                if
                                :: src != id ->
                                        if
                                        :: msgType == IAmCentral ->
                                                announceReceived = 0 ;
                                                negoReceived = 0 ;

                                                if
                                                :: my_nego_list ?? [eval(src),
announceReceived, negoReceived] ->
                                                        my_nego_list ?? src,
announceReceived, negoReceived
                                                :: else
                                                fi ;

                                                if
                                                :: !(announceReceived == 1 &&
negoReceived == 1) ->
                                                        sendMessage( CentralNego,
Unicast, rank, id, src ) ;
                                                        centralNego = 1 ;
                                                        reset( waitDelay ) ;
                                                        set( centralNegoMax, 30 ) ;
                                                :: else
                                                fi ;

                                                announceReceived = 1 ;
                                                negoReceived = 0 ;
                                                my_nego_list ! src, announceReceived,
negoReceived ;

                                                announceReceived = 0 ;
                                                negoReceived = 0
```

```
                                        :: msgType == LeaderElect ->
                                                sendMessage( IAmCentral, Unicast, 0, id,
255 ) ;
                                                checkRanklist( src )

                                        :: else -> checkRanklist( src )
                                        fi
                                :: else
                                fi

                        :: transType == Unicast ->
                                if
                                :: msgType == BackupAssignAck ->
                                        if
                                        :: src == lastBackup ->
                                                assignBackup = 0 ;
                                                backupAssigned = 1 ;
                                                waitBackupReply = 0 ;
                                                sendHelloBackup = 1 ;
                                                set( backupDelay, 15 ) ;

                                                sendRanklist( lastBackup )
                                        :: else ->
                                                sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
                                                checkRanklist( src )
                                        fi

                                :: msgType == CentralAssignAck ->
                                        if
                                        :: src == lastAppointed ->
                                                if
                                                :: backupAssigned == 1 ->
                                                        sendMessage( BackupCancel,
Unicast, 0, id, lastBackup ) ;
                                                :: else
                                                fi ;

                                                sendRanklist( lastAppointed ) ;

                                                clearVariables() ;
                                                lastCentral = src ;

                                                if
                                                :: id == 0 -> rightAssigned == 0
                                                :: else
                                                fi ;

                                                mode = 4 ;
                                                nr_leader-- ;
                                                leader[id] = 0 ;
                                                break
                                        :: else
                                        fi

                                :: msgType == CentralNego ->
```

```
                                                announceReceived = 0 ;
                                                negoReceived = 0 ;

                                                if
                                                :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                                                        my_nego_list ?? src, announceReceived,
negoReceived
                                                :: else
                                                fi ;

                                                if
                                                :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                                        sendMessage( CentralNego, Unicast, rank,
id, src ) ;
                                                :: else
                                                fi ;

                                                negoReceived = 1 ;
                                                my_nego_list ! src, announceReceived,
negoReceived ;


                                                announceReceived = 0 ;
                                                negoReceived = 0 ;

                                                if
                                                :: msg > rank ->
                                                        printf( "MSC: LOST\n" ) ;
                                                        if
                                                        :: backupAssigned == 1 ->
                                                                sendMessage( BackupCancel,
Unicast, 0, id, lastBackup )
                                                        :: else
                                                        fi ;

                                                        clearVariables() ;
                                                        clearRanklist() ;

                                                        if
                                                        :: id == 0 -> rightAssigned = 0
                                                        :: else
                                                        fi ;

                                                        mode = 3 ;
                                                        nr_leader-- ;
                                                        leader[id] = 0 ;
                                                        break
                                                :: else ->
                                                        if
                                                        :: centralNego == 0 ->
                                                                centralNego = 1 ;
                                                                reset( waitDelay ) ;
                                                                set( centralNegoMax, 30 )
                                                        :: else
                                                        fi
```

```
                                        fi ;

                                        if
                                        :: backupAssigned == 1 && msg > backupRank ->
                                                sendMessage( BackupCancel, Unicast, 0, id,
lastBackup ) ;
                                                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;

                                                lastBackup = src ;
                                                backupRank = msg ;
                                                assignBackup = 0 ;
                                                backupAssigned = 0 ;
                                                waitBackupReply = 1 ;
                                                sendHelloBackup = 0 ;
                                                set( backupDelay, 15 )
                                        :: else
                                        fi ;

                                        if
                                        :: ranklistEmpty == 1 ->
                                                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;

                                                lastBackup = src ;
                                                backupRank = msg ;
                                                assignBackup = 0 ;
                                                waitBackupReply = 1 ;
                                                sendHelloBackup = 0 ;
                                                set( backupDelay, 30 ) ;
                                                ranklistEmpty = 0
                                        :: else
                                        fi ;

                                        updateRanklist( msg, src ) ;
                                        sendRankEntry( src, lastBackup )

                                :: msgType == HelloCentral ->
                                        if
                                        :: src == lastBackup ->
                                                sendHelloBackup = 1 ;
                                                set( backupDelay, 15 )
                                        :: else ->
                                                sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
                                                checkRanklist( src )
                                        fi

                                :: msgType == MoreCentrals ->
                                        announceReceived = 0 ;
                                        negoReceived = 0 ;

                                        if
                                        :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
```

```
                                        my_nego_list ?? src, announceReceived,
negoReceived
                                    :: else
                                    fi ;

                                    if
                                    :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                        sendMessage( CentralNego, Unicast, rank,
id, msg ) ;
                                    :: else
                                    fi ;

                                    announceReceived = 0 ;
                                    negoReceived = 0 ;

                                    if
                                    :: src != lastBackup ->
                                        sendMessage( BackupCancel, Unicast, rank,
id, src )
                                    :: else
                                    fi ;

                                    checkRanklist( src )

                                :: msgType == MyResReply ->
                                    if
                                    :: msg > rank ->
                                        sendMessage( NewCentralAssign, Unicast,
rank, id, src ) ;

                                        lastAppointed = src ;
                                        waitAppointAck = 1 ;
                                        set( centralAppointDelay, 30 )

                                    :: else
                                    fi ;

                                    if
                                    :: backupAssigned == 1 && msg > backupRank ->
                                        sendMessage( BackupCancel, Unicast, 0, id,
lastBackup ) ;
                                        sendMessage( BackupAssign, Unicast, 0, id,
src ) ;

                                        lastBackup = src ;
                                        backupRank = msg ;
                                        assignBackup = 0 ;
                                        backupAssigned = 0 ;
                                        waitBackupReply = 1 ;
                                        sendHelloBackup = 0 ;
                                        set( backupDelay, 15 )
                                    :: else
                                    fi ;

                                    if
```

```
                                        :: ranklistEmpty == 1 ->
                                                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;

                                                lastBackup = src ;
                                                backupRank = msg ;
                                                assignBackup = 0 ;
                                                backupAssigned = 0 ;
                                                waitBackupReply = 1 ;
                                                sendHelloBackup = 0 ;
                                                set( backupDelay, 30 ) ;
                                                ranklistEmpty = 0
                                        :: else
                                        fi ;

                                        updateRanklist( msg, src ) ;
                                        sendRankEntry( src, lastBackup )

                                :: msgType == RankEntryAdd ->
                                        if
                                        :: src == lastBackup ->
                                                addRankEntry( msg )
                                        :: else
                                        fi

                                :: msgType == SrvSearch ->
                                        /*sendMessage( SrvSearchReply, Unicast, 0, id, src )
;*/
                                        checkRanklist( src )

                                :: else -> checkRanklist( src )
                                fi
                        fi ;

                        clearMail()
                }

        /*** Announcements ***/
        :: centralNego == 0 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;
                set( waitDelay, 30 )

        /*** Backup assignment ***/
        :: centralNego == 0 && assignBackup == 1 ->
                delay( nodeDelay, 1 ) ;
                findHighest() ;

                if
                :: anID == 255 ->
                        printf( "MSC: Empty ranklist\n" ) ;
                        ranklistEmpty = 1 ;
                        assignBackup = 0 ;
                        backupAssigned = 0 ;
                        waitBackupReply = 0 ;
                        lastBackup = 255 ;
```

```
                backupRank = 0 ;
                sendHelloBackup = 0
        :: else ->
                lastBackup = anID ;
                backupRank = aRank ;

                sendMessage( BackupAssign, Unicast, 0, id, anID ) ;
                assignBackup = 0 ;
                backupAssigned = 0 ;
                waitBackupReply = 1 ;
                sendHelloBackup = 0 ;
                set( backupDelay, 30 ) ;
        fi

:: waitBackupReply == 1 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( BackupAssign, Unicast, 0, id, lastBackup ) ;
        assignBackup = 0 ;
        backupAssigned = 0 ;
        waitBackupReply = 2 ;
        sendHelloBackup = 0 ;
        set( backupDelay, 30 ) ;

:: waitBackupReply == 2 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        printf("MSC: Remove\n") ;
        removeEntry( lastBackup ) ;
        lastBackup = 255 ;
        backupRank = 0 ;
        assignBackup = 1 ;
        backupAssigned = 0 ;
        waitBackupReply = 0 ;
        sendHelloBackup = 0 ;

/*** Backup polling ***/
:: sendHelloBackup == 1 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
        sendHelloBackup = 2 ;
        set( backupDelay, 30 )
:: sendHelloBackup == 2 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
        sendHelloBackup = 3 ;
        set( backupDelay, 30 )
:: sendHelloBackup == 3 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        removeEntry( lastBackup ) ;
        lastBackup = 255 ;
        backupRank = 0 ;
        assignBackup = 1 ;
        backupAssigned = 0 ;
        waitBackupReply = 0 ;
        sendHelloBackup = 0 ;

/*** Central negotiation max timeout ***/
```

```
		:: centralNego == 1 && centralNegoMax.val <= 0 ->
			if
			:: from_mail_to_i ?? [eval(IAmCentral), transType, msg, src, dest] ||
				from_mail_to_i ?? [eval(CentralNego), transType, msg, src, dest] ->
					set( centralNegoMax, 30 )
			:: else ->
				centralNego = 0 ;
				reset( waitDelay ) ;

				do
				:: my_nego_list ? _, _, _
				:: empty( my_nego_list ) -> break
				od
			fi

		/*** Central appointment ***/
		:: waitAppointAck == 1 && centralAppointDelay.val <= 0 ->
			sendMessage( NewCentralAssign, Unicast, rank, id, lastAppointed ) ;
			waitAppointAck = 2 ;
			set( centralAppointDelay, 30 ) ;
		:: waitAppointAck == 2 && centralAppointDelay.val <= 0 ->
			waitAppointAck = 0 ;
			lastAppointed = 255
		od ;

:: mode == 3 ->
		printf("MSC: NORMAL\n") ;

		if
		:: startup == 1 ->
			if
			:: leader[0] == 1 -> lastCentral = 0
			:: leader[1] == 1 -> lastCentral = 1
			:: leader[2] == 1 -> lastCentral = 2
			:: leader[3] == 1 -> lastCentral = 3
			:: else -> printf("MSC: ERROR\n")
			fi ;

			set( waitDelay, 30 ) ;
			startup = 0
		:: else -> lastCentral = 255
		fi ;

		do
		:: from_mail_to_i ? msgType, transType, msg, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: msgType == IAmCentral ->
					lastCentral = src ;

				:: msgType == BackupAssign ->
					if
					:: lastCentral != 255 && src != lastCentral ->
```

```
                                                sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
                                                sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                        :: else ->
                                                sendMessage( BackupAssignAck, Unicast, 0, id, src )
;

                                                clearVariables() ;
                                                lastCentral = src ;

                                                mode = 4 ;
                                                break
                                        fi

                                :: msgType == NewCentralAssign ->
                                        sendMessage( CentralAssignAck, Unicast, 0, id, src ) ;

                                        clearVariables() ;
                                        lastBackup = src ;
                                        backupRank = msg ;
                                        sendHelloBackup = 1 ;
                                        assignBackup = 0 ;
                                        backupAssigned = 1 ;

                                        mode = 2 ;
                                        leader[id] = 1 ;
                                        break

                                :: msgType == MyResRqst ->
                                        sendMessage( MyResReply, Unicast, rank, id, src ) ;

                                :: else
                                        fi ;

                                        clearMail()
                        }

                :: lastCentral != 255 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( SrvSearch, Unicast, 0, id, lastCentral ) ;
                        set( waitDelay, 30 )
                od

        :: mode == 4 ->
                printf("MSC: BACKUP\n") ;
                nr_backup++ ;
                set( waitDelay, 60 ) ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
```

```
//                              :: msgType == IAmCentral ->
//                                      if
//                                      :: src != lastCentral ->
//                                              sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
//                                              sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
//                                      :: else
//                                      fi

                                        :: msgType == BackupAssign ->
                                                if
                                                :: src == lastCentral ->
                                                        sendMessage( BackupAssignAck, Unicast, 0, id, src )
                                                :: else ->
                                                        sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
                                                        sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                                fi

                                        :: msgType == BackupCancel ->
                                                clearRanklist() ;
                                                clearVariables() ;

                                                mode = 3 ;
                                                nr_backup-- ;
                                                break

                                        :: msgType == HelloDevice ->
                                                if
                                                :: src == lastCentral ->
                                                        sendMessage( HelloCentral, Unicast, 0, id,
lastCentral ) ;

                                                        sendHelloCentral = 0 ;
                                                        set( waitDelay, 60 )
                                                :: else //->
//                                                      sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
//                                                      sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                                fi

                                        :: msgType == RankEntryAdd ->
                                                if
                                                :: src == lastCentral ->
                                                        addRankEntry( msg )
                                                :: else //->
//                                                      sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
//                                                      sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                                fi

                                        :: else
```

```
                                        fi ;

                                        clearMail()
                                }

                        /*** Central polling ***/
                        :: sendHelloCentral == 0 && waitDelay.val <= 0 ->
                                delay( nodeDelay, 1 ) ;
                                sendMessage( HelloCentral, Unicast, 0, id, lastCentral ) ;
                                sendHelloCentral = 1 ;
                                set( waitDelay, 60 )
                        :: sendHelloCentral == 1 && waitDelay.val <= 0 ->
                                delay( nodeDelay, 1 ) ;

                                clearVariables() ;

                                mode = 2 ;
                                nr_backup-- ;
                                leader[id] = 1 ;
                                break
                        od
        od
}

init
{
        atomic
        {
                if
                :: leader[0] = 1 ; nr_leader++
                :: false
                fi ;

                if
                :: leader[1] = 1 ; nr_leader++
                :: false
                fi ;

                if
                :: leader[2] = 1 ; nr_leader++
                :: false
                fi ;

                if
                :: leader[3] = 1 ; nr_leader++
                :: false
                fi ;

                if
                :: nr_leader == 0 ->
                        if
                        :: leader[0] = 1
                        :: leader[1] = 1
                        :: leader[2] = 1
                        :: leader[3] = 1
                        fi
```

```
                :: else
                fi ;

                nr_leader = 0 ;

                run Node(0, 10) ;
                run Node(1, 4) ;
                run Node(2, 8) ;
                run Node(3, 6)
        }
}



// Backup Assignment with message loss – version 1
// Ceryen Tan, Vasughi Sundramoorthy
// 8/20 - Clear buffer error
// 8/17 - Messaging renaming
// 8/16 - Cleanup attempt

#include "dtime.h"

#define N 3
#define MAX_CH 6
#define MAX_LOSS 1

#define Unicast 0
#define Broadcast 1

mtype = {
                MyResource, IAmCentral, CentralNego, LeaderElect, MoreCentrals,
                NewCentralAssign, CentralAssignAck,
                BackupAssign, BackupAssignAck, HelloCentral, HelloDevice, BackupCancel,
RankEntryAdd,
                MyResRqst, MyResReply, SrvRegRqst,
                SrvSearch, SrvSearchReply
        } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
chan nego_list[N] = [N] of {byte, bit, bit } ;

byte nr_leader = 0 ;
byte nr_backup = 0 ;
byte rightAssigned = 0 ;
byte ranklistSize = 0 ;

byte lossCounters[17] ;

typedef rankEntry
{
        byte id, rank ;
} ;

inline addRankEntry( anID )
{
        atomic
        {
```

```
                    if
                    :: anID != id ->
                            if
                            :: anID == 0 -> updateRanklist( 10, 0 )
                            :: anID == 1 -> updateRanklist( 4, 1 )
                            :: anID == 2 -> updateRanklist( 8, 2 )
                            fi
                    :: else
                    fi
            }
}

inline clearBuffer()
{
            atomic
            {
                    do
                    :: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
                    :: empty(from_mail_to_i) -> break
                    od
            }
}

inline clearMail()
{
            msgType = 0 ;
            transType = 0 ;
            msg = 0 ;
            src = 0 ;
            dest = 0 ;
            type = 0
}

inline checkRanklist( anID )
{
            atomic
            {
                    findInRanklist( anID ) ;

                    if
                    :: tempLoc == 255 -> sendMessage( MyResRqst, Unicast, 0, id, anID )
                    :: else -> tempLoc = 0
                    fi
            }
}

inline clearRanklist()
{
            atomic
            {
                    counter = 0 ;
                    do
                    :: counter < N -> rankList[counter].id = 255 ; rankList[counter].rank = 255 ; counter++
                    :: counter == N -> counter = 0 ; break
                    od ;
```

```
                    if
                    :: id == 0 -> ranklistSize = 0
                    :: else
                    fi
            }
}

inline clearVariables()
{
            announceReceived = 0 ;
            negoReceived = 0 ;

            mode = 0 ;
            waitMode = 0 ;
            centralNego = 0 ;

            waitAnnounce = 0 ;
            startSearch = 0;
            sendSrvSearch = 0 ;
            waitSrvSearchReply = 0 ;

            waitAppointAck = 0 ;
            lastAppointed = 255 ;

            lastCentral = 255 ;
            lastBackup = 255 ;
            sendHelloBackup = 0 ;
            sendHelloCentral = 0 ;

            assignBackup = 0 ;
            backupAssigned = 0 ;
            waitBackupReply = 0 ;
            attemptedAssign = 0 ;

            backupRank = 0 ;
            ranklistEmpty = 0 ;

            reset( waitDelay ) ;
            reset( nodeDelay ) ;
            reset( backupDelay ) ;
            reset( centralNegoMax ) ;
            reset( centralAppointDelay ) ;

            do
            :: nempty( my_nego_list ) -> my_nego_list ? _, _, _
            :: empty( my_nego_list ) -> break
            od
}

inline findHighest()
{
            atomic
            {
                    counter = 0 ;
                    aRank = 0 ;
                    anID = 255 ;
```

```
                do
                :: counter < N ->
                        if
                        :: rankList[counter].rank != 255 && rankList[counter].rank > aRank ->
                                aRank = rankList[counter].rank ;
                                anID = rankList[counter].id
                        :: else
                        fi ;
                        counter++
                :: counter == N -> counter = 0 ; break
                od
        }
}

inline findInRanklist( anID )
{
        atomic
        {
                tempLoc = 255 ;
                counter = 0 ;
                do
                :: counter < N ->
                        if
                        :: rankList[counter].id == anID -> tempLoc = counter ; break
                        :: else
                        fi ;
                        counter++
                :: counter == N -> counter = 0 ; break
                od
        }
}

inline getType( msgType )
{
        if
        :: msgType == MyResource -> type = 0
        :: msgType == IAmCentral -> type = 1
        :: msgType == CentralNego -> type = 2
        :: msgType == LeaderElect -> type = 3
        :: msgType == MoreCentrals -> type = 4
        :: msgType == NewCentralAssign -> type = 5
        :: msgType == CentralAssignAck -> type = 6
        :: msgType == BackupAssign -> type = 7
        :: msgType == BackupAssignAck -> type = 8
        :: msgType == HelloCentral -> type = 9
        :: msgType == HelloDevice -> type = 10
        :: msgType == BackupCancel -> type = 11
        :: msgType == MyResRqst -> type = 12
        :: msgType == MyResReply -> type = 13
        :: msgType == SrvRegRqst -> type = 14
        :: msgType == SrvSearch -> type = 15
        :: msgType == SrvSearchReply -> type = 16
        :: else -> type = 255
        fi
}
```

```
inline removeEntry( anID )
{
	atomic
	{
		counter = 0 ;

		do
		:: counter < N ->
			if
			:: rankList[counter].id == anID ->
				rankList[counter].id = 255 ;
				rankList[counter].rank = 255 ;

				if
				:: id == 0 -> ranklistSize--
				:: else
				fi
			:: else
			fi ;
			counter++
		:: counter == N -> counter = 0 ; break
		od
	}
}

inline sendRankEntry( toSend, dest )
{
	atomic
	{
		if
		:: lastBackup != 255 && backupAssigned == 1 -> sendMessage( RankEntryAdd,
Unicast, toSend, id, dest )
		:: else
		fi
	}
}

inline sendRanklist( dest )
{
	atomic
	{
		if
		:: lastBackup != 255 && backupAssigned == 1 ->
			counter = 0 ;

			do
			:: counter < N ->
				if
				:: rankList[counter].id != 255 -> sendMessage( RankEntryAdd,
Unicast, counter, id, dest )
				:: else
				fi ;
				counter++

			:: counter >= N -> break
```

```
                od ;

                counter = 0
            :: else
            fi
        }
}

inline updateRanklist( aRank, anID )
{
        atomic
        {
            counter = 0 ;
            tempLoc = 255 ;

            do
            :: counter < N ->
                if
                :: rankList[counter].id == anID -> tempLoc = counter ; break
                :: else ->
                    if
                    :: tempLoc == 255 &&
                        rankList[counter].rank == 255 &&
                        rankList[counter].id == 255 ->
                        tempLoc = counter
                    :: else -> counter++
                    fi
                fi ;
            :: counter == N -> break
            od ;

            if
            :: tempLoc == 255 -> printf("MSC: Full Ranklist\n")
            :: else ->
                if
                :: id == 0 && rankList[tempLoc].id != anID -> ranklistSize++
                :: else
                fi ;

                rankList[tempLoc].rank = aRank ;
                rankList[tempLoc].id = anID
            fi ;

            counter = 0 ;
            tempLoc = 0 ;
        }
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
        getType( msgType ) ;

        atomic
        {
            if
            :: transType == Unicast ->
```

```
                        if
                        :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                        :: else ->
                                if
                                :: from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                                :: true -> printf( "MSC: UNICAST: %d, %d, %d, %d\n", type, aMsg,
aSrc, dest ) ; lossCounters[type]++
                                fi
                        fi ;
                :: transType == Broadcast ->
                        dest = 0 ;

                        do
                        :: dest < N ->
                                if
                                :: dest != id ->
                                        if
                                        :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                                from_mail[dest] ! msgType, transType, aMsg, aSrc,
dest
                                        :: else ->
                                                if
                                                :: from_mail[dest] ! msgType, transType, aMsg,
aSrc, dest
                                                :: true -> printf( "MSC: BROADCAST: %d, %d, %d,
%d\n", type, aMsg, aSrc, dest ) ;
                                                        lossCounters[type]++
                                                fi
                                        fi ;
                                :: else
                                fi ;

                                dest++
                        :: dest >= N -> dest = 0 ; break
                        od ;
                fi ;

                type = 0
        }
}

proctype Node(byte id, rank)
{
        /* Mail */
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;


        /* Messages */
        mtype msgType ;
        bit transType ;
        byte msg ;
        byte src ;
        byte dest ;
        byte type ;
```

```
        /* Central negotiation table */
        chan my_nego_list = nego_list[id] ;
        xr my_nego_list ;
        xs my_nego_list ;

        bit announceReceived ;
        bit negoReceived ;

        /* Ranklist */
        rankEntry rankList[N] ;
        byte counter ;
        byte tempLoc ;
        byte aRank, anID ;

        /* Central flags */
        byte mode ;
        byte waitMode ;
        bit centralNego ;

        /* Normal flags */
        byte waitAnnounce ;
        byte startSearch;
        byte sendSrvSearch ;
        byte waitSrvSearchReply ;

        /* Central appoint */
        byte waitAppointAck ;
        byte lastAppointed ;

        /* Backup */
        byte lastCentral ;
        byte lastBackup ;
        byte sendHelloBackup ;
        byte sendHelloCentral ;

        byte assignBackup ;
        byte backupAssigned ;
        byte waitBackupReply ;
        bit attemptedAssign ;

        byte backupRank ;
        byte ranklistEmpty ;

        /* Timers */
        timer waitDelay ;
        timer nodeDelay ;
        timer backupDelay ;
        timer centralNegoMax ;
        timer centralAppointDelay ;

        printf("MSC: START\n") ;
        atomic { clearBuffer() } ;

start:
        atomic { clearRanklist() } ;
```

```
	do
	:: mode == 0 ->                         /***** Startup *****/
		sendMessage( MyResource, Broadcast, rank, id, 255 ) ;
		delay( nodeDelay, 1 ) ;
		nr_leader++ ;
		mode++

	:: mode == 1 ->                         /***** Listen mode *****/
		set( waitDelay, 30 ) ;

		do
		:: waitDelay.val > 0 ->
			if
			:: from_mail_to_i ? msgType, transType, msg, src, dest ->
				delay( nodeDelay, 1 ) ;

				atomic
				{
					if
					:: transType == Broadcast ->
						if
						:: src != id ->
							if
							:: msgType == MyResource ->
								printf( "MSC: %d\n", msg ) ;
								if
								:: msg > rank ->
									printf( "MSC: LOST\n" ) ;

									clearRanklist() ;
									clearVariables() ;
									waitAnnounce = 1 ;

									mode = 3 ;
									nr_leader-- ;
									break
								:: else -> updateRanklist( msg, src )
; sendRankEntry( src, lastBackup )
								fi

							:: else
							fi
						:: else
						fi

					:: transType == Unicast ->
						if
						:: msgType == BackupAssign ->
							printf( "MSC: LOST\n" ) ;
							sendMessage( BackupAssignAck, Unicast,
0, id, src ) ;

							clearRanklist() ;
							clearVariables() ;
							lastCentral = src ;
```

```
                                                mode = 4 ;
                                                nr_leader-- ;
                                                nr_backup++ ;
                                                break

                                        :: msgType == NewCentralAssign ->
                                                printf( "MSC: WON\n" ) ;
                                                sendMessage( CentralAssignAck, Unicast,
0, id, src ) ;

                                                clearVariables() ;
                                                lastBackup = src ;
                                                backupRank = msg ;
                                                sendHelloBackup = 1 ;
                                                assignBackup = 0 ;
                                                backupAssigned = 1 ;

                                                mode = 2 ;
                                                break

                                        :: msgType == SrvRegRqst ->
                                                printf( "MSC: LOST\n" ) ;

                                                clearRanklist() ;
                                                clearVariables() ;

                                                mode = 3 ;
                                                nr_leader-- ;
                                                break

                                        :: else
                                                fi
                                fi ;

                                clearMail()
                        }

                :: empty(from_mail_to_i) -> delay( nodeDelay, 1 )
                fi

        :: waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                printf( "MSC: WON\n") ;

                mode = 2 ;
                break
        od

:: mode == 2 ->                                 /***** Central mode *****/
        printf("MSC: CENTRAL\n");

        if
        :: id == 0 -> rightAssigned = 1
        :: else
        fi ;
```

```
end1:           do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast ->
                                        if
                                        :: src != id ->
                                                if
                                                :: msgType == IAmCentral ->
                                                        announceReceived = 0 ;
                                                        negoReceived = 0 ;

                                                        if
                                                        :: my_nego_list ?? [eval(src),
announceReceived, negoReceived] ->
                                                                my_nego_list ?? src,
announceReceived, negoReceived
                                                        :: else
                                                        fi ;

                                                        if
                                                        :: !(announceReceived == 1 &&
negoReceived == 1) ->
                                                                sendMessage( CentralNego,
Unicast, rank, id, src ) ;

                                                                centralNego = 1 ;
                                                                waitMode = 2 ;
                                                                reset( waitDelay ) ;
                                                                set( centralNegoMax, 30 ) ;
                                                        :: else
                                                        fi ;

                                                        announceReceived = 1 ;
                                                        negoReceived = 0 ;
                                                        my_nego_list ! src, announceReceived,
negoReceived ;

                                                        announceReceived = 0 ;
                                                        negoReceived = 0

                                                :: msgType == LeaderElect ->
                                                        sendMessage( IAmCentral, Broadcast, 0, id,
255 ) ;
                                                        checkRanklist( src )

                                                :: msgType == MyResource ->
                                                        if
                                                        :: msg > rank ->
                                                                sendMessage( NewCentralAssign,
Unicast, 0, id, src ) ;
```

```
                                                waitAppointAck = 1 ;
                                                lastAppointed = src ;
                                                set( centralAppointDelay, 30 )

                                        :: msg <= rank ->
                                                sendMessage( SrvRegRqst,
Unicast, 0, id, src )
                                        fi ;

                                        if
                                        :: msg > backupRank && (backupAssigned
== 1 || ranklistEmpty == 1) ->
                                                if
                                                :: lastBackup != 255 ->
                                                        sendMessage(
BackupCancel, Unicast, 0, id, lastBackup ) ;
                                                :: else
                                                fi ;

                                                sendMessage( BackupAssign,
Unicast, 0, id, src ) ;
                                                lastBackup = src ;
                                                backupRank = msg ;
                                                waitBackupReply = 1 ;
                                                backupAssigned = 0 ;
                                                sendHelloBackup = 0 ;
                                                assignBackup = 0 ;
                                                attemptedAssign = 1 ;
                                                set( backupDelay, 30 )
                                        :: else
                                        fi ;

                                                ranklistEmpty = 0 ;
                                                updateRanklist( msg, src ) ;
                                                sendRankEntry( src, lastBackup )

                                        :: else -> checkRanklist( src )
                                        fi
                                :: else
                                fi

                        :: transType == Unicast ->
                                if
                                :: msgType == BackupAssignAck ->
                                        if
                                        :: src == lastBackup ->
                                                sendRanklist( lastBackup ) ;

                                                backupAssigned = 1 ;
                                                waitBackupReply = 0 ;
                                                sendHelloBackup = 1 ;
                                                assignBackup = 0 ;
                                                set( backupDelay, 30 )
                                        :: else ->
                                                sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
```

```
                                        checkRanklist( src )
                                fi

                        :: msgType == CentralAssignAck ->
                                if
                                :: src == lastAppointed ->
                                        if
                                        :: backupAssigned == 1 ->
                                                sendMessage( BackupCancel,
Unicast, 0, id, lastBackup)
                                        :: else
                                        fi ;

                                        sendRanklist( lastAppointed ) ;

                                        clearVariables() ;
                                        lastCentral = src ;

                                        if
                                        :: id == 0 -> rightAssigned = 0
                                        :: else
                                        fi ;

                                        mode = 4 ;
                                        nr_leader-- ;
                                        break
                                :: else
                                fi

                        :: msgType == CentralNego ->
                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                                        my_nego_list ?? src, announceReceived,
negoReceived
                                :: else
                                fi ;

                                if
                                :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                        sendMessage( CentralNego, Unicast, rank,
id, src ) ;
                                :: else
                                fi ;

                                negoReceived = 1 ;
                                my_nego_list ! src, announceReceived,
negoReceived ;

                                announceReceived = 0 ;
                                negoReceived = 0 ;
```

```
                                        if
                                        :: msg > rank ->
                                                printf("MSC: LOST\n") ;
                                                if
                                                :: backupAssigned == 1 ->
                                                        sendMessage( BackupCancel,
Unicast, 0, id, lastBackup )
                                                :: else
                                                fi ;

                                                clearRanklist() ;
                                                clearVariables() ;
                                                waitAnnounce = 1 ;

                                                if
                                                :: id == 0 -> rightAssigned = 0
                                                :: else
                                                fi ;

                                                mode = 3 ;
                                                nr_leader-- ;
                                                break
                                        :: else ->
                                                if
                                                :: centralNego == 0 ->
                                                        centralNego = 1 ;
                                                        waitMode = 2 ;
                                                        reset( waitDelay ) ;
                                                        set( centralNegoMax, 30 )
                                                :: else
                                                fi ;

                                                updateRanklist( msg, src ) ;
                                                sendRankEntry( src, lastBackup )
                                        fi ;

                                        if
                                        :: msg > backupRank && (backupAssigned == 1 ||
ranklistEmpty == 1) ->
                                                if
                                                :: backupAssigned == 255 ->
                                                        sendMessage( BackupCancel,
Unicast, 0, id, lastBackup ) ;
                                                :: else
                                                fi ;

                                                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;
                                                lastBackup = src ;
                                                backupRank = msg ;
                                                waitBackupReply = 1 ;
                                                backupAssigned = 0 ;
                                                sendHelloBackup = 0 ;
                                                assignBackup = 0 ;
                                                attemptedAssign = 1 ;
                                                set( backupDelay, 30 )
```

```
                        :: else
                        fi ;

                        ranklistEmpty = 0

                :: msgType == HelloCentral ->
                        if
                        :: src == lastBackup ->
                                sendHelloBackup = 1 ;
                                set( backupDelay, 30 )
                        :: else ->
                                sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
                                checkRanklist( src )
                        fi

                :: msgType == MoreCentrals ->
                        announceReceived = 0 ;
                        negoReceived = 0 ;

                        if
                        :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                                my_nego_list ?? src, announceReceived,
negoReceived
                        :: else
                        fi ;

                        if
                        :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                sendMessage( CentralNego, Unicast, rank,
id, msg ) ;
                        :: else
                        fi ;

                        announceReceived = 0 ;
                        negoReceived = 0 ;

                        if
                        :: src != lastBackup ->
                                sendMessage( BackupCancel, Unicast, 0, id,
src )
                        :: else ->
                        fi ;

                        checkRanklist( src )

                :: msgType == MyResReply ->
                        if
                        :: msg > rank ->
                                sendMessage( NewCentralAssign, Unicast,
rank, id, src ) ;

                                waitAppointAck = 1 ;
                                lastAppointed = src ;
```

```
                                        set( centralAppointDelay, 30 )

                                    :: else

                                    fi ;

                                    if
                                    :: msg > backupRank && (backupAssigned == 1 ||
ranklistEmpty == 1) ->
                                        if
                                        :: backupAssigned == 1 ->
                                            sendMessage( BackupCancel,
Unicast, 0, id, lastBackup ) ;
                                        :: else
                                        fi ;

                                        sendMessage( BackupAssign, Unicast, 0, id,
src ) ;
                                        lastBackup = src ;
                                        backupRank = msg ;
                                        waitBackupReply = 1 ;
                                        sendHelloBackup = 0 ;
                                        assignBackup = 0 ;
                                        attemptedAssign = 0 ;
                                        set( backupDelay, 30 )
                                    :: else
                                    fi ;

                                    ranklistEmpty = 0 ;
                                    updateRanklist( msg, src ) ;
                                    sendRankEntry( src, lastBackup )

                                :: msgType == SrvSearch ->
                                    sendMessage( SrvSearchReply, Unicast, 0, id, src ) ;
                                    checkRanklist( src )

                                :: msgType == RankEntryAdd ->
                                    if
                                    :: src == lastBackup ->
                                        addRankEntry( msg )
                                    :: else
                                    fi

                                :: else -> checkRanklist( src )
                                fi
                        fi ;

                        clearMail()
                }

            /*** Announce modes ***/
            :: waitMode == 0 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;
                waitMode++ ;
                set( waitDelay, 30 )
```

```
:: waitMode == 1 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;
        waitMode++ ;

        if
        :: attemptedAssign == 0 -> assignBackup = 1 ; attemptedAssign = 1
        :: else
        fi ;

        set( waitDelay, 30 )
:: centralNego == 0 && waitMode == 2 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;

        if
        :: attemptedAssign == 0 -> assignBackup = 1 ; attemptedAssign = 1
        :: else
        fi ;

        set( waitDelay, 30 )

/*** Central Negotiation ***/
:: centralNego == 1 && centralNegoMax.val <= 0 ->
        if
        :: from_mail_to_i ?? [eval(IAmCentral), transType, msg, src, dest] ||
                from_mail_to_i ?? [eval(CentralNego), transType, msg, src, dest] ->
                        set( centralNegoMax, 30 )
        :: else ->
                centralNego = 0 ;               /* Do an announce with 0 first */
                waitMode = 2 ;
                reset( waitDelay ) ;

                do
                :: nempty( my_nego_list ) -> my_nego_list ? _, _, _
                :: empty( my_nego_list ) -> break
                od
        fi

/*** Backup assignment ***/
:: centralNego == 0 && assignBackup == 1 ->
        delay( nodeDelay, 1 ) ;

        atomic
        {
                findHighest() ;

                if
                :: anID == 255 ->
                        printf("MSC: Empty List\n") ;
                        assignBackup = 0 ;
                        lastBackup = 255 ;
                        backupRank = 0 ;
                        ranklistEmpty = 1 ;
                        sendHelloBackup = 0 ;
                        waitBackupReply = 0 ;
```

```
                        backupAssigned = 0 ;
                :: else ->
                        printf("MSC: Backup %d\n", anID) ;
                        sendMessage( BackupAssign, Unicast, 0, id, anID ) ;
                        lastBackup = anID ;
                        backupRank = aRank ;
                        assignBackup = 0 ;
                        waitBackupReply = 1 ;
                        sendHelloBackup = 0 ;
                        backupAssigned = 0 ;
                        set( backupDelay, 15 )
                fi
        }
:: waitBackupReply == 1 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( BackupAssign, Unicast, 0, id, lastBackup ) ;
        waitBackupReply = 2 ;
        assignBackup = 0 ;
        backupAssigned = 0 ;
        sendHelloBackup = 0 ;
        set( backupDelay, 15 )
:: waitBackupReply == 2 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        removeEntry( lastBackup ) ;
        waitBackupReply = 0 ;
        lastBackup = 255 ;
        backupRank = 0 ;
        assignBackup = 1 ;
        backupAssigned = 0 ;
        sendHelloBackup = 0

/*** Backup Polling ***/
:: sendHelloBackup == 1 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
        sendHelloBackup = 2 ;
        set( backupDelay, 30 )
:: sendHelloBackup == 2 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
        sendHelloBackup = 3 ;
        set( backupDelay, 30 )
:: sendHelloBackup == 3 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        removeEntry( lastBackup ) ;
        lastBackup = 255 ;
        backupRank = 0 ;
        assignBackup = 1 ;
        backupAssigned = 0 ;
        sendHelloBackup = 0 ;
        waitBackupReply = 0

/*** Central Appointment ***/
:: waitAppointAck == 1 && centralAppointDelay.val <= 0 ->
        sendMessage( NewCentralAssign, Unicast, rank, id, lastAppointed ) ;
        waitAppointAck = 2 ;
```

```
                        set( centralAppointDelay, 30 )
                :: waitAppointAck == 2 && centralAppointDelay.val <= 0 ->
                        waitAppointAck = 0 ;
                        lastAppointed = 255
                od

        :: mode == 3 ->                         /***** Normal mode *****/
                printf("MSC: NORMAL\n");
                sendSrvSearch = 1 ;
                if
                :: waitAnnounce == 1 -> set( waitDelay, 60 )
                :: else
                fi ;

end2:           do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast ->
                                        if
                                        :: src != id ->
                                                if
                                                :: msgType == IAmCentral ->
                                                        waitAnnounce = 0 ;

                                                        if
                                                        :: lastCentral == 255 || src != lastCentral ->
                                                                waitSrvSearchReply = 0
                                                        :: else
                                                        fi;

                                                        lastCentral = src ;

                                                :: msgType == LeaderElect ->
                                                        waitAnnounce = 2 ;
                                                        set( waitDelay, 30 )

                                                :: else
                                                fi
                                        :: else
                                        fi

                                :: transType == Unicast ->
                                        if
                                        :: msgType == BackupAssign ->
                                                if
                                                :: lastCentral != 255 && src != lastCentral ->
                                                        sendMessage( MoreCentrals, Unicast, src,
id, lastCentral ) ;
                                                        sendMessage( MoreCentrals, Unicast,
lastCentral, id, src )
                                                :: else ->
```

```
                                                sendMessage( BackupAssignAck, Unicast,
0, id, src ) ;

                                                clearVariables() ;
                                                lastCentral = src ;

                                                mode = 4 ;
                                                nr_backup++ ;
                                                break
                                        fi

                                :: msgType == NewCentralAssign ->
                                        sendMessage( CentralAssignAck, Unicast, 0, id, src )
;

                                                clearVariables() ;
                                                lastBackup = src ;
                                                backupRank = msg ;
                                                sendHelloBackup = 1 ;
                                                assignBackup = 0 ;
                                                backupAssigned = 1 ;

                                                mode = 2 ;
                                                nr_leader++ ;
                                                break

                                :: msgType == MyResRqst ->
                                        sendMessage( MyResReply, Unicast, rank, id, src );
                                        sendSrvSearch=1;

                                :: msgType == SrvSearchReply ->
                                        waitSrvSearchReply = 0 ;
                                        sendSrvSearch = 2 ;
                                        set( waitDelay, 60 )

                                :: else
                                fi
                        fi ;

                        clearMail()
                }

        /*** Initial wait for announcement ***/
        :: waitAnnounce == 1 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( LeaderElect, Broadcast, 0, id, 255 ) ;
                waitAnnounce = 2 ;
                set( waitDelay, 30 )
        :: waitAnnounce == 2 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                goto start

        /*** Service Searching ***/
        :: waitAnnounce == 0 && sendSrvSearch == 1 && waitSrvSearchReply == 0 ->
                delay( nodeDelay, 1 ) ;
                set( waitDelay, 60 ) ;
```

```
                        sendSrvSearch = 2;
                :: sendSrvSearch == 2 && waitDelay.val <= 0 && waitSrvSearchReply == 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( SrvSearch, Unicast, 0, id, lastCentral ) ;
                        waitSrvSearchReply = 1 ;
                        set( waitDelay, 60 )
                :: waitSrvSearchReply == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( LeaderElect, Broadcast, 0, id, 255 ) ;
                        waitAnnounce = 2 ;
                        waitSrvSearchReply = 0 ;
                        set( waitDelay, 30 )
                od
        :: mode == 4 ->
                printf("MSC: BACKUP\n") ;
                set( waitDelay, 60 ) ;

end3:           do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast -> skip
                                :: transType == Unicast ->
                                        if
//                                      :: msgType == IAmCentral ->
//                                              if
//                                              :: src != lastCentral ->
//                                                      sendMessage( MoreCentrals, Unicast, src,
id, lastCentral ) ;
//                                                      sendMessage( MoreCentrals, Unicast,
lastCentral, id, src )
//                                              :: else
//                                              fi

                                        :: msgType == BackupAssign ->
                                                if
                                                :: src == lastCentral ->
                                                        sendMessage( BackupAssignAck, Unicast,
0, id, lastCentral ) ;
                                                :: else ->
                                                        sendMessage( MoreCentrals, Unicast, src,
id, lastCentral ) ;
                                                        sendMessage( MoreCentrals, Unicast,
lastCentral, id, src )
                                                fi

                                        :: msgType == BackupCancel ->
                                                clearRanklist() ;
                                                clearVariables() ;

                                                mode = 3 ;
                                                nr_backup-- ;
                                                break
```

```
                                        :: msgType == HelloDevice ->
                                                if
                                                :: src == lastCentral ->
                                                        sendMessage( HelloCentral, Unicast, 0, id,
src ) ;

                                                        sendHelloCentral = 1 ;
                                                        set( waitDelay, 30 )
                                                :: else //->
//                                                      sendMessage( MoreCentrals, Unicast, src,
id, lastCentral ) ;
//                                                      sendMessage( MoreCentrals, Unicast,
lastCentral, id, src )
                                                fi

                                        :: msgType == RankEntryAdd ->
                                                if
                                                :: src == lastCentral ->
                                                        addRankEntry( msg )
                                                :: else //->
//                                                      sendMessage( MoreCentrals, Unicast, src,
id, lastCentral ) ;
//                                                      sendMessage( MoreCentrals, Unicast,
lastCentral, id, src )
                                                fi

                                        :: else
                                                fi
                                fi ;

                                clearMail()
                        }

                /*** Central Polling ***/
                :: sendHelloCentral == 0 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( HelloCentral, Unicast, 0, id, lastCentral ) ;
                        sendHelloCentral = 2 ;
                        set( waitDelay, 30 )
                :: sendHelloCentral == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( HelloCentral, Unicast, 0, id, lastCentral ) ;
                        sendHelloCentral = 2 ;
                        set( waitDelay, 30 )
                :: sendHelloCentral == 2 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;

                        clearVariables() ;

                        mode = 2 ;
                        nr_leader++ ;
                        nr_backup -- ;
                        break
                od
        od
}
```

```
init
{
	atomic
	{
		run Node(0, 10) ;
		run Node(1, 4) ;
		run Node(2, 8)
	}
}


// Backup Assignment with message loss – version 2
// Ceryen Tan, Vasughi Sundramoorthy
// 8/20 - Clear buffer error
// 8/17 - Cleanup attempt, Message renaming

#include "dtime.h"

#define N 3
#define MAX_CH 5
#define MAX_LOSS 1

#define Unicast 0
#define Broadcast 1

mtype = {
		IAmCentral, CentralNego, MoreCentrals,
		NewCentralAssign, CentralAssignAck,
		BackupAssign, BackupAssignAck, HelloCentral, HelloDevice, BackupCancel,
RankEntryAdd,
		MyResRqst, MyResReply,
		SrvSearch, SrvSearchReply
	} ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
chan nego_list[N] = [N] of {byte, bit, bit } ;

byte nr_leader = 0 ;
byte nr_backup = 0 ;
byte rightAssigned = 0 ;
byte ranklistSize = 0 ;

byte lossCounters[13] ;

typedef rankEntry
{
	byte id, rank ;
} ;

inline addRankEntry( anID )
{
	atomic
	{
		if
		:: anID != id ->
			if
```

```
                        :: anID == 0 -> updateRanklist( 10, 0 )
                        :: anID == 1 -> updateRanklist( 4, 1 )
                        :: anID == 2 -> updateRanklist( 8, 2 )
                        fi
                :: else
                fi
        }
}

inline checkRanklist( anID )
{
        atomic
        {
                findInRanklist( anID ) ;

                if
                :: tempLoc == 255 -> sendMessage( MyResRqst, Unicast, 0, id, anID )
                :: else -> tempLoc = 0
                fi
        }
}

inline clearBuffer()
{
        atomic
        {
                do
                :: nempty(from_mail_to_i) -> from_mail_to_i ? _, _, _, _, _
                :: empty(from_mail_to_i) -> break
                od
        }
}

inline clearMail()
{
        msgType = 0 ;
        transType = 0 ;
        msg = 0 ;
        src = 0 ;
        dest = 0 ;
        type = 0
}

inline clearRanklist()
{
        atomic
        {
                counter = 0 ;
                do
                :: counter < N -> rankList[counter].id = 255 ; rankList[counter].rank = 255 ; counter++
                :: counter == N -> counter = 0 ; break
                od ;

                if
                :: id == 0 -> ranklistSize = 0
                :: else
```

```
                fi
        }
}

inline clearVariables()
{
        waitAppointAck = 0 ;
        lastAppointed = 255 ;

        lastCentral = 255 ;
        lastBackup = 255 ;
        sendHelloBackup = 0 ;
        sendHelloCentral = 0 ;

        assignBackup = 0 ;
        backupAssigned = 0 ;
        waitBackupReply = 0 ;

        backupRank = 0 ;
        ranklistEmpty = 0 ;

        centralNego = 0 ;

        reset( waitDelay ) ;
        reset( nodeDelay ) ;
        reset( backupDelay ) ;
        reset( centralNegoMax ) ;
        reset( centralAppointDelay ) ;

        do
        :: nempty( my_nego_list ) -> my_nego_list ? _, _, _
        :: empty( my_nego_list ) -> break
        od
}

inline findHighest()
{
        atomic
        {
                counter = 0 ;
                aRank = 0 ;
                anID = 255 ;

                do
                :: counter < N ->
                        if
                        //unchanged- changed rankList[counter].rank to id
                        :: rankList[counter].rank != 255 && rankList[counter].rank > aRank ->
                                aRank = rankList[counter].rank ;
                                anID = rankList[counter].id
                        :: else
                        fi ;
                        counter++
                :: counter == N -> counter = 0 ; break
                od
        }
```

```
}

inline findInRanklist( anID )
{
	atomic
	{
		tempLoc = 255 ;
		counter = 0 ;
		do
		:: counter < N ->
			if
			:: rankList[counter].id == anID -> tempLoc = counter ; break
			:: else
			fi ;
			counter++
		:: counter == N -> break
		od ;
		counter = 0
	}
}

inline getType( msgType )
{
	if
	:: msgType == IAmCentral -> type = 0
	:: msgType == CentralNego -> type = 1
	:: msgType == NewCentralAssign -> type = 2
	:: msgType == BackupAssign -> type = 3
	:: msgType == HelloCentral -> type = 4
	:: msgType == HelloDevice -> type = 5
	:: msgType == BackupCancel -> type = 6
	:: msgType == MyResRqst -> type = 7
	:: msgType == MyResReply -> type = 8
	:: msgType == SrvSearch -> type = 9
	:: msgType == SrvSearchReply -> type = 10
	:: msgType == BackupAssignAck -> type = 11
	:: msgType == MoreCentrals -> type = 12
	:: else -> type = 255
	fi
}

inline removeEntry( anID )
{
	atomic
	{
		counter = 0 ;

		do
		:: counter < N ->
			if
			:: rankList[counter].id == anID ->
				rankList[counter].id = 255 ;
				rankList[counter].rank = 255 ;

				if
				:: id == 0 -> ranklistSize--
```

```
                                        :: else
                                        fi
                                :: else
                                fi ;
                                counter++
                        :: counter == N -> counter = 0 ; break
                        od
        }
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
        getType( msgType ) ;

        atomic
        {
                if
                :: transType == Unicast ->
                        if
                        :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                        :: else ->
                                if
                                :: from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                                :: true -> printf( "MSC: UNICAST: %d, %d, %d\n", aMsg, aSrc, dest )
; lossCounters[type]++
                                fi
                        fi ;
                :: transType == Broadcast ->
                        dest = 0 ;

                        do
                        :: dest < N ->
                                if
                                :: dest != id ->
                                        if
                                        :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                                from_mail[dest] ! msgType, transType, aMsg, aSrc,
dest
                                        :: else ->
                                                if
                                                :: from_mail[dest] ! msgType, transType, aMsg,
aSrc, dest

                                                :: true -> printf( "MSC: BROADCAST: %d, %d,
%d\n", aMsg, aSrc, dest ) ;
                                                        lossCounters[type]++
                                                fi
                                        fi ;
                                :: else
                                fi ;

                                dest++
                        :: dest >= N -> dest = 0 ; break
                        od ;
                fi ;
```

```
                type = 0
        }
}

inline sendRankEntry( toSend, dest )
{
        atomic
        {
                if
                :: lastBackup != 255 && backupAssigned == 1 -> sendMessage( RankEntryAdd,
Unicast, toSend, id, dest )
                :: else
                fi
        }
}

inline sendRanklist( dest )
{
        atomic
        {
                if
                :: lastBackup != 255 && backupAssigned == 1 ->
                        counter = 0 ;

                        do
                        :: counter < N ->
                                if
                                :: rankList[counter].id != 255 -> sendMessage( RankEntryAdd,
Unicast, counter, id, dest )
                                :: else
                                fi ;
                                counter++

                        :: counter >= N -> break
                        od ;

                        counter = 0
                :: else
                fi
        }
}

inline updateRanklist( aRank, anID )
{
        atomic
        {
                counter = 0 ;
                tempLoc = 255 ;

                do
                :: counter < N ->
                        if
                        :: rankList[counter].id == anID -> tempLoc = counter ; break
                        :: else ->
                                if
                                :: tempLoc == 255 &&
```

```
                                        rankList[counter].rank == 255 &&
                                        rankList[counter].id == 255 ->
                                        tempLoc = counter
                                :: else -> counter++
                                fi
                        fi ;
                :: counter == N -> break
                od ;

                if
                :: tempLoc == 255 -> printf("MSC: Full Ranklist\n")
                :: else ->
                        if
                        :: id == 0 && rankList[tempLoc].id != anID -> ranklistSize++
                        :: else
                        fi ;

                        rankList[tempLoc].rank = aRank ;
                        rankList[tempLoc].id = anID
                fi ;

                counter = 0 ;
                tempLoc = 0
        }
}

proctype Node( byte id, rank, mode )
{
        /* Mail */
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* Messages */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;
        byte type ;

        /* Central negotiation table */
        chan my_nego_list = nego_list[id] ;
        xr my_nego_list ;
        xs my_nego_list ;

        bit announceReceived ;
        bit negoReceived ;

        /* Ranklist */
        rankEntry rankList[N] ;
        byte counter ;
        byte tempLoc ;
        byte aRank, anID ;

        /* Central appoint */
        byte waitAppointAck ;
```

```
	byte lastAppointed ;

	/* Backup */
	byte lastCentral ;
	byte lastBackup ;
	byte sendHelloBackup ;
	bit sendHelloCentral ;

	bit assignBackup ;
	bit backupAssigned ;
	byte waitBackupReply ;

	byte backupRank ;
	bit ranklistEmpty ;

	/* Central negotation */
	bit centralNego ;

	/* Timers */
	timer waitDelay ;
	timer nodeDelay ;
	timer backupDelay ;
	timer centralNegoMax ;
	timer centralAppointDelay ;

	printf("MSC: START\n") ;
	atomic { clearBuffer() } ;

start:
	atomic
	{
		clearRanklist() ;
		clearVariables()
	} ;

	do
	:: mode == 2 ->
		printf("MSC: LEADER\n") ;

		if
		:: id == 0 -> rightAssigned = 1
		:: else
		fi ;

		assignBackup = 1 ;
		set( waitDelay, 30 ) ;

		nr_leader++ ;

		do
		:: from_mail_to_i ? msgType, transType, msg, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
```

```
                    :: transType == Broadcast ->
                        if
                        :: src != id ->
                            if
                            :: msgType == IAmCentral ->
                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: my_nego_list ?? [eval(src),
announceReceived, negoReceived] ->
                                    my_nego_list ?? src,
announceReceived, negoReceived
                                :: else
                                fi ;

                                if
                                :: !(announceReceived == 1 &&
negoReceived == 1) ->
                                    sendMessage( CentralNego,
Unicast, rank, id, src ) ;
                                    centralNego = 1 ;
                                    reset( waitDelay ) ;
                                    set( centralNegoMax, 30 ) ;
                                :: else
                                fi ;

                                announceReceived = 1 ;
                                negoReceived = 0 ;
                                my_nego_list ! src, announceReceived,
negoReceived ;

                                announceReceived = 0 ;
                                negoReceived = 0

                            :: else -> checkRanklist( src )
                            fi
                        :: else
                        fi

                    :: transType == Unicast ->
                        if
                        :: msgType == BackupAssignAck ->
                            if
                            :: src == lastBackup ->
                                sendRanklist( lastAppointed ) ;

                                assignBackup = 0 ;
                                backupAssigned = 1 ;
                                waitBackupReply = 0 ;
                                sendHelloBackup = 1 ;
                                set( backupDelay, 15 )
                            :: else ->
                                sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
                                checkRanklist( src )
```

```
                        fi

                    :: msgType == CentralAssignAck ->
                        if
                        :: src == lastAppointed ->
                            if
                            :: backupAssigned == 1 ->
                                sendMessage( BackupCancel,
Unicast, 0, id, lastBackup ) ;
                            :: else
                            fi ;

                            sendRanklist( lastAppointed ) ;

                            clearVariables() ;
                            lastCentral = src ;

                            if
                            :: id == 0 -> rightAssigned = 0
                            :: else
                            fi ;

                            mode = 4 ;
                            nr_leader-- ;
                            nr_backup++ ;
                            break
                        :: else
                        fi

                    :: msgType == CentralNego ->
                        announceReceived = 0 ;
                        negoReceived = 0 ;

                        if
                        :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                            my_nego_list ?? src, announceReceived,
negoReceived
                        :: else
                        fi ;

                        if
                        :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                            sendMessage( CentralNego, Unicast, rank,
id, src ) ;
                        :: else
                        fi ;

                        negoReceived = 1 ;
                        my_nego_list ! src, announceReceived,
negoReceived ;

                        announceReceived = 0 ;
                        negoReceived = 0 ;
```

```
                                        if
                                        :: msg > rank ->
                                                printf("MSC: LOST\n") ;
                                                if
                                                :: backupAssigned == 1 ->
                                                        sendMessage( BackupCancel,
Unicast, 0, id, lastBackup )
                                                :: else
                                                fi ;

                                                clearRanklist() ;
                                                clearVariables() ;

                                                if
                                                :: id == 0 -> rightAssigned = 0
                                                :: else
                                                fi ;

                                                mode = 3 ;
                                                nr_leader-- ;
                                                break
                                        :: else ->
                                                if
                                                :: centralNego == 0 ->
                                                        centralNego = 1 ;
                                                        reset( waitDelay ) ;
                                                        set( centralNegoMax, 30 )
                                                :: else
                                                fi
                                        fi ;

                                        if
                                        :: backupAssigned == 1 && msg > backupRank ->
                                                sendMessage( BackupCancel, Unicast, 0, id,
lastBackup ) ;
                                                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;
                                                lastBackup = src ;
                                                backupRank = msg ;

                                                assignBackup = 0 ;
                                                backupAssigned = 0 ;
                                                waitBackupReply = 1 ;
                                                sendHelloBackup = 0 ;
                                                set( backupDelay, 15 )
                                        :: else
                                        fi ;

                                        if
                                        :: ranklistEmpty == 1 ->
                                                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;

                                                lastBackup = src ;
                                                backupRank = msg ;
                                                assignBackup = 0 ;
```

```
                                        waitBackupReply = 1 ;
                                        backupAssigned = 0 ;
                                        sendHelloBackup = 0 ;
                                        set( backupDelay, 15 ) ;
                                        ranklistEmpty = 0
                                :: else
                                fi ;

                                updateRanklist( msg, src ) ;
                                sendRankEntry( src, lastBackup )

                        :: msgType == HelloCentral ->
                                if
                                :: src == lastBackup ->
                                        sendHelloBackup = 1 ;
                                        set( backupDelay, 15 )
                                :: else ->
                                        sendMessage( BackupCancel, Unicast, 0, id,
src ) ;
                                        checkRanklist( src )
                                fi

                        :: msgType == MoreCentrals ->
                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                                        my_nego_list ?? src, announceReceived,
negoReceived
                                :: else
                                fi ;

                                if
                                :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                        sendMessage( CentralNego, Unicast, rank,
id, msg ) ;
                                :: else
                                fi ;

                                announceReceived = 0 ;
                                negoReceived = 0 ;

                                if
                                :: src != lastBackup ->
                                        sendMessage( BackupCancel, Unicast, 0, id,
src )
                                :: else
                                fi ;

                                checkRanklist( src )

                        :: msgType == MyResReply ->
                                if
```

```
                                        :: msg > rank ->
                                                sendMessage( NewCentralAssign, Unicast,
rank, id, src ) ;

                                                waitAppointAck = 1 ;
                                                lastAppointed = src ;
                                                set( centralAppointDelay, 30 )

                                        :: else
                                        fi ;

                                        if
                                        :: backupAssigned == 1 && msg > backupRank ->
                                                sendMessage( BackupCancel, Unicast, 0, id,
lastBackup ) ;
                                                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;
                                                lastBackup = src ;
                                                backupRank = msg ;

                                                assignBackup = 0 ;
                                                backupAssigned = 0 ;
                                                waitBackupReply = 1 ;
                                                sendHelloBackup = 0 ; //unch- changed acc
to 0812
                                                set( backupDelay, 15 )
                                        :: else
                                        fi ;

                                        if
                                        :: ranklistEmpty == 1 ->
                                                sendMessage( BackupAssign, Unicast, 0, id,
src ) ;

                                                lastBackup = src ;
                                                backupRank = msg ;
                                                assignBackup = 0 ;
                                                backupAssigned = 0 ;
                                                waitBackupReply = 1 ;
                                                sendHelloBackup = 0 ;
                                                set( backupDelay, 15 ) ;
                                                ranklistEmpty = 0
                                        :: else
                                        fi ;

                                        updateRanklist( msg, src ) ;
                                        sendRankEntry( src, lastBackup )

                                :: msgType == SrvSearch ->
                                        /*sendMessage( SrvSearchReply, Unicast, 0, id, src )
;*/
                                        checkRanklist( src )

                                :: msgType == RankEntryAdd ->
                                        if
                                        :: src == lastBackup ->
```

```
                                        addRankEntry( msg )
                                :: else
                                fi

                        :: else -> checkRanklist( src )
                        fi
                fi ;

                clearMail()
        }

/*** Announcements ***/
:: centralNego == 0 && waitDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;
        set( waitDelay, 30 )

/*** Backup Assignment ***/
:: centralNego == 0 && assignBackup == 1 ->
        delay( nodeDelay, 1 ) ;
        findHighest() ;

        if
        :: anID == 255 ->
                printf( "MSC: Empty ranklist\n" ) ;
                ranklistEmpty = 1 ;
                assignBackup = 0 ;
                backupAssigned = 0 ;
                waitBackupReply = 0 ;
                lastBackup = 255 ;
                backupRank = 0 ;
                sendHelloBackup = 0
        :: else ->
                lastBackup = anID ;
                backupRank = aRank ;
                sendMessage( BackupAssign, Unicast, 0, id, anID ) ;
                assignBackup = 0 ;
                backupAssigned = 0 ;
                waitBackupReply = 1 ;
                sendHelloBackup = 0 ; //unch-changed acc to 0812
                set( backupDelay, 30 ) ;
        fi

:: waitBackupReply == 1 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        sendMessage( BackupAssign, Unicast, 0, id, lastBackup ) ;
        assignBackup = 0 ;
        backupAssigned = 0 ;
        waitBackupReply = 2 ;
        sendHelloBackup = 0 ;
        set( backupDelay, 30 ) ;

:: waitBackupReply == 2 && backupDelay.val <= 0 ->
        delay( nodeDelay, 1 ) ;
        printf("MSC: Remove\n") ;
        removeEntry( lastBackup ) ;
```

```
                lastBackup = 255 ;
                backupRank = 0 ;
                assignBackup = 1 ;
                backupAssigned = 0 ;
                waitBackupReply = 0 ;
                sendHelloBackup = 0 ;

        /*** Backup polling ***/
        :: sendHelloBackup == 1 && backupDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
                sendHelloBackup = 2 ;
                set( backupDelay, 30 )
        :: sendHelloBackup == 2 && backupDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( HelloDevice, Unicast, 0, id, lastBackup ) ;
                sendHelloBackup = 3 ;
                set( backupDelay, 30 )
        :: sendHelloBackup == 3 && backupDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                removeEntry( lastBackup ) ;
                lastBackup = 255 ;
                backupRank = 0 ;
                assignBackup = 1 ;
                backupAssigned = 0 ;
                waitBackupReply = 0 ;
                sendHelloBackup = 0 ;

        /*** Central negotation ***/
        :: centralNego == 1 && centralNegoMax.val <= 0 ->
                if
                :: from_mail_to_i ?? [eval(IAmCentral), transType, msg, src, dest] ||
                        from_mail_to_i ?? [eval(CentralNego), transType, msg, src, dest] ->
                                set( centralNegoMax, 30 )
                :: else ->
                        centralNego = 0 ;
                        reset( waitDelay ) ;

                        do
                        :: my_nego_list ? _, _, _
                        :: empty( my_nego_list ) -> break
                        od
                fi

        /*** Central appointment ***/
        :: waitAppointAck == 1 && centralAppointDelay.val <= 0 ->
                sendMessage( NewCentralAssign, Unicast, rank, id, lastAppointed ) ;
                waitAppointAck = 2 ;
                set( centralAppointDelay, 30 )
        :: waitAppointAck == 2 && centralAppointDelay.val <= 0 ->
                waitAppointAck = 0 ;
                lastAppointed = 255
        od ;

:: mode == 3 ->
        printf("MSC: NORMAL\n") ;
```

```
		do
		:: from_mail_to_i ? msgType, transType, msg, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: msgType == IAmCentral ->
					lastCentral = src ;

				:: msgType == BackupAssign ->
					if
					:: lastCentral != 255 && src != lastCentral ->
						sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
						sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
					:: else ->
						sendMessage( BackupAssignAck, Unicast, 0, id, src )
;

						clearVariables() ;
						lastCentral = src ;

						mode = 4 ;
						nr_backup++ ;
						break
					fi

				:: msgType == NewCentralAssign ->
					sendMessage( CentralAssignAck, Unicast, 0, id, src ) ;

					clearVariables() ;
					lastBackup = src ;
					backupRank = msg ;
					sendHelloBackup = 1 ;
					assignBackup = 0 ;
					backupAssigned = 1 ;

					mode = 2 ;
					break

				:: msgType == MyResRqst ->
					sendMessage( MyResReply, Unicast, rank, id, src ) ;

				:: else
				fi ;

				clearMail()
			}

		/*** Periodic searching ***/
		:: lastCentral != 255 && waitDelay.val <= 0 ->
			delay( nodeDelay, 1 ) ;
			sendMessage( SrvSearch, Unicast, 0, id, lastCentral ) ;
```

```
                        set( waitDelay, 30 )
                od

        :: mode == 4 ->
                printf("MSC: BACKUP\n") ;
                set( waitDelay, 60 ) ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
//                              :: msgType == IAmCentral ->
//                                      if
//                                      :: src != lastCentral ->
//                                              sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
//                                              sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
//                                      :: else
//                                      fi

                                :: msgType == BackupAssign ->
                                        if
                                        :: src == lastCentral ->
                                                sendMessage( BackupAssignAck, Unicast, 0, id,
lastCentral )
                                        :: else ->
                                                sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
                                                sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                        fi

                                :: msgType == BackupCancel ->
                                        clearRanklist() ;
                                        clearVariables() ;

                                        mode = 3 ;
                                        nr_backup-- ;
                                        break

                                :: msgType == HelloDevice ->
                                        if
                                        :: src == lastCentral ->
                                                sendMessage( HelloCentral, Unicast, 0, id,
lastCentral ) ;

                                                sendHelloCentral = 0 ;
                                                set( waitDelay, 60 )
                                        :: else //->
//                                              sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
```

```
//                                              sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                        fi

                                    :: msgType == RankEntryAdd ->
                                        if
                                        :: src == lastCentral ->
                                                addRankEntry( msg )
                                        :: else //->
//                                              sendMessage( MoreCentrals, Unicast, src, id,
lastCentral ) ;
//                                              sendMessage( MoreCentrals, Unicast, lastCentral, id,
src )
                                        fi

                                    :: else
                                    fi ;

                                    clearMail()
                            }

                    /*** Central polling ***/
                    :: sendHelloCentral == 0 && waitDelay.val <= 0 ->
                            delay( nodeDelay, 1 ) ;
                            sendMessage( HelloCentral, Unicast, 0, id, lastCentral ) ;
                            sendHelloCentral = 1 ;
                            set( waitDelay, 60 )
                    :: sendHelloCentral == 1 && waitDelay.val <= 0 ->
                            delay( nodeDelay, 1 ) ;

                            clearVariables() ;

                            mode = 2 ;
                            nr_backup-- ;
                            break
                    od
            od
}

init
{
            atomic
            {
                    run Node(0, 10, 2) ;
                    run Node(1, 4, 3) ;
                    run Node(2, 8, 3)
            }
}
```

```
// Registration for 300D with message loss
// Ceryen Tan, Vasughi Sundramoorthy
// 8/17 - Cleanup attempt, Message renaming

#include "dtime.h"

#define N 3
#define MAX_CH 5
#define MAX_LOSS 1

#define Unicast 0
#define Broadcast 1

mtype =
	{
		MyResource, IAmCentral, LeaderElect, CentralNego,
		SrvRegRqst, SrvReg, UnsolSrvReg, SrvRegRenew, SrvRegReply,
		HelloDevice, HelloCentral,
		SrvFound, SrvNotFound, SrvSearch, NewCentralAssign
	} ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;
chan nego_list[N] = [N] of {byte, bit, bit } ;

bit found300D = 1 ;
byte lossCounters[15] ;

inline clearMail()
{
	msgType = 0 ;
	transType = 0 ;
	msg = 0 ;
	src = 0 ;
	dest = 0
}

inline getType( msgType )
{
	byte type ;

	if
	:: msgType == IAmCentral -> type = 0
	:: msgType == SrvRegRqst -> type = 1
	:: msgType == SrvReg -> type = 2
	:: msgType == UnsolSrvReg -> type = 3
	:: msgType == HelloDevice -> type = 4
	:: msgType == HelloCentral -> type = 5
	:: msgType == SrvFound -> type = 6
	:: msgType == SrvNotFound -> type = 7
	:: msgType == SrvSearch -> type = 8
	:: msgType == LeaderElect -> type = 9
	:: msgType == MyResource -> type = 10
	:: msgType == SrvRegReply -> type = 11
	:: msgType == NewCentralAssign -> type = 12
	:: msgType == CentralNego -> type = 13
	:: else -> type = 255
	fi
```

```
}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
	getType( msgType ) ;

	atomic
	{
		if
		:: transType == Unicast ->
			if
			:: type == 255 || lossCounters[type] >= MAX_LOSS ->
				from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
			:: else ->
				if
				:: from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
				:: true -> printf( "MSC: UNICAST: %d, %d, %d\n", aMsg, aSrc, dest )
; lossCounters[type]++
				fi
			fi ;
		:: transType == Broadcast ->
			dest = 0 ;

			do
			:: dest < N ->
				if
				:: dest != id ->
					if
					:: type == 255 || lossCounters[type] >= MAX_LOSS ->
						from_mail[dest] ! msgType, transType, aMsg, aSrc,
dest
					:: else ->
						if
						:: from_mail[dest] ! msgType, transType, aMsg,
aSrc, dest
						:: true -> printf( "MSC: BROADCAST: %d, %d,
%d\n", aMsg, aSrc, dest ) ;
							lossCounters[type]++
						fi
					fi ;
				:: else
				fi ;

				dest++
			:: dest >= N -> dest = 0 ; break
			od ;
		fi ;

		type = 0
	}
}

proctype Node300D( byte id, mode, rank )
{
	chan from_mail_to_i = from_mail[id] ;
	chan my_nego_list = nego_list[id] ;
```

```
xr from_mail_to_i ;
xr my_nego_list ;
xs my_nego_list ;

/* MESSAGES */
mtype msgType ;
byte transType ;
byte msg ;
byte src ;
byte dest ;

byte lastCentral ;
byte announceMode ;
bit waitSrvRegRqst ;
bit sendRegRenew ;
byte waitSrvRegReply ;
bit centralNego ;

bit announceReceived ;
bit negoReceived ;

bit reg300D ;
bit wait300D ;

timer regRenewDelay ;
timer nodeDelay ;
timer centralNegoMax ;
timer waitDelay ;
timer talk300D ;

do
:: mode == 1 ->
        printf("MSC: LISTEN\n") ;
        set( waitDelay, 30 ) ;
        sendMessage( MyResource, Broadcast, rank, id, 255 ) ;

        do
        :: waitDelay.val > 0 ->
                if
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast -> skip
                                :: transType == Unicast ->
                                        if
                                        :: msgType == NewCentralAssign ->       /*
SHOULD NOT HAPPEN */
                                                mode = 2 ;
                                                break

                                        :: msgType == SrvRegRqst ->
                                                sendMessage( SrvReg, Unicast, 1, id, src ) ;
                                                mode = 3 ;
```

```
                                                        break

                                                :: else
                                                fi
                                        fi ;

                                        clearMail()
                                }

                        :: empty( from_mail_to_i ) -> delay( nodeDelay, 1 )
                        fi

                :: waitDelay.val <= 0 ->
                        mode = 2 ;
                        break
                od

        :: mode == 2 ->
                printf("MSC: NORMAL\n") ;
                lastCentral = 255 ;
                sendRegRenew = 0 ;
                waitSrvRegRqst = 0 ;
                reset( regRenewDelay ) ;
                reset( waitDelay ) ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast ->
                                        if
                                        :: src != id ->
                                                if
                                                :: msgType == IAmCentral ->
                                                        if
                                                        :: msg == 1 || msg == 2 ->
                                                                if
                                                                :: src != lastCentral ->
                                                                        sendMessage(
UnsolSrvReg, Unicast, 1, id, src ) ;
                                                                        waitSrvRegReply = 1 ;
                                                                        set( waitDelay, 15 ) ;
                                                                        lastCentral = src
                                                                :: else
                                                                fi
                                                        :: msg == 3 ->
                                                                lastCentral = src ;
                                                                waitSrvRegRqst = 1 ;
                                                                waitSrvRegReply = 0 ;
                                                                set( waitDelay, 15 )
                                                        :: else
                                                        fi
                                                :: else
```

```
                                        fi
                                :: else
                                fi

                        :: transType == Unicast ->
                                if
                                :: msgType == SrvRegRqst ->
                                        sendMessage( SrvReg, Unicast, 1, id, src )
                                        lastCentral = src ;
                                        sendRegRenew = 1 ;
                                        set( regRenewDelay, 15 ) ;
                                        waitSrvRegRqst = 0

                                :: msgType == SrvRegReply ->
                                        waitSrvRegReply = 0 ;
                                        reset( waitDelay ) ;
                                        sendRegRenew = 1 ;
                                        set( regRenewDelay, 15 )

                                :: else
                                fi
                        fi ;

                        clearMail()
                }

/*** Renewal ***/
:: sendRegRenew == 1 && regRenewDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( SrvRegRenew, Unicast, 0, id, lastCentral ) ;
                set( regRenewDelay, 15 )

/*** Initial wait for service reg request ***/
:: waitSrvRegRqst == 1 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( UnsolSrvReg, Unicast, 1, id, lastCentral ) ;
                waitSrvRegRqst = 0 ;
                waitSrvRegReply = 1 ;
                set( waitDelay, 15 )

/*** Wait for reply to unsolicited service registration ***/
:: waitSrvRegReply == 1 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( LeaderElect, Broadcast, 0, id, 255 ) ;
                lastCentral = 255 ;
                sendRegRenew = 0 ;
                reset( regRenewDelay ) ;
                waitSrvRegReply = 2 ;
                set( waitDelay, 15 )

:: waitSrvRegReply == 2 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                mode = 1 ;

                waitSrvRegReply = 0 ;
                sendRegRenew = 0 ;
```

```
                        lastCentral = 255 ;
                        reset( regRenewDelay ) ;
                        reset( waitDelay ) ;

                        break
                od

        :: mode == 3 ->
                printf("MSC: LEADER\n") ;
                announceMode = 0 ;
                reg300D = 0 ;
                wait300D = 0 ;
                reset( waitDelay ) ;
                reset( talk300D ) ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast ->
                                        if
                                        :: src != id ->
                                                if
                                                :: msgType == IAmCentral ->
                                                        announceReceived = 0 ;
                                                        negoReceived = 0 ;

                                                        if
                                                        :: my_nego_list ?? [eval(src),
announceReceived, negoReceived] ->
                                                                my_nego_list ?? src,
announceReceived, negoReceived
                                                        :: else
                                                        fi ;

                                                        if
                                                        :: !(announceReceived == 1 &&
negoReceived == 1) ->
                                                                sendMessage( CentralNego,
Unicast, rank, id, src ) ;

                                                                centralNego = 1 ;
                                                                announceMode = 3 ;
                                                                reset( waitDelay ) ;
                                                                set( centralNegoMax, 30 ) ;
                                                        :: else
                                                        fi ;

                                                        announceReceived = 1 ;
                                                        negoReceived = 0 ;
                                                        my_nego_list ! src, announceReceived,
negoReceived ;

                                                        announceReceived = 0 ;
```

```
                                        negoReceived = 0

                                    :: msgType == MyResource ->
                                        if
                                        :: msg < rank -> sendMessage( SrvRegRqst,
Unicast, 0, id, src )
                                        :: else ->                          /*
SHOULD NOT HAPPEN */
                                            sendMessage( NewCentralAssign,
Unicast, 0, id, src ) ;
                                            mode = 2 ;

                                            announceMode = 0 ;
                                            reg300D = 0 ;
                                            wait300D = 0 ;
                                            centralNego = 0 ;
                                            reset( waitDelay ) ;
                                            reset( talk300D ) ;
                                            reset( centralNegoMax ) ;

                                            do
                                            :: my_nego_list ? _, _, _
                                            :: empty( my_nego_list ) -> break
                                            od ;

                                            break
                                        fi
                                    :: else
                                        fi
                                :: else
                                fi

                            :: transType == Unicast ->
                                if
                                :: msgType == CentralNego ->
                                    announceReceived = 0 ;
                                    negoReceived = 0 ;

                                    if
                                    :: my_nego_list ?? [eval(src), announceReceived,
negoReceived] ->
                                        my_nego_list ?? src, announceReceived,
negoReceived
                                    :: else
                                    fi ;

                                    if
                                    :: ( announceReceived == 1 && negoReceived == 0 )
|| ( announceReceived == 0 ) ->
                                        sendMessage( CentralNego, Unicast, rank,
id, src ) ;
                                    :: else
                                    fi ;

                                    negoReceived = 1 ;
```

```
                                        my_nego_list ! src, announceReceived,
negoReceived ;

                                        announceReceived = 0 ;
                                        negoReceived = 0 ;

                                        if
                                        :: msg > rank ->
                                                mode = 2 ;

                                                announceMode = 0 ;
                                                reg300D = 0 ;
                                                wait300D = 0 ;
                                                centralNego = 0 ;
                                                reset( waitDelay ) ;
                                                reset( talk300D ) ;
                                                reset( centralNegoMax ) ;

                                                do
                                                :: my_nego_list ? _, _, _
                                                :: empty( my_nego_list ) -> break
                                                od ;

                                                break
                                        :: else ->
                                                if
                                                :: centralNego == 0 ->
                                                        centralNego = 1 ;
                                                        announceMode = 3 ;
                                                        reset( waitDelay ) ;
                                                        set( centralNegoMax, 30 )
                                                :: else
                                                fi
                                        fi

                                :: msgType == SrvReg ->
                                        if
                                        :: msg == 1 ->
                                                printf("MSC: REG 300D\n");
                                                reg300D = 1 ;
                                                set( talk300D, 30 ) ;

                                                if
                                                :: wait300D == 1 -> sendMessage(
SrvFound, Unicast, 1, id, 2 ) ; wait300D = 0
                                                :: else
                                                fi
                                        :: else
                                        fi

                                :: msgType == UnsolSrvReg ->
                                        sendMessage( SrvRegReply, Unicast, 0, id, src ) ;

                                        if
                                        :: msg == 1 ->
                                                printf("MSC: REG 300D\n");
```

```
                                                reg300D = 1 ;
                                                set( talk300D, 30 ) ;

                                                if
                                                :: wait300D == 1 -> sendMessage(
SrvFound, Unicast, 1, id, 2 ) ; wait300D = 0
                                                :: else
                                                fi
                                        :: else
                                        fi

                                :: msgType == SrvRegRenew ->
                                        if
                                        :: reg300D == 1 -> printf("MSC: 300D RENEW\n") ;
set( talk300D, 60 )
                                        :: else -> sendMessage( SrvRegRqst, Unicast, 0, id,
src )
                                        fi

                                :: msgType == SrvSearch ->
                                        if
                                        :: msg == 1 && reg300D == 1 -> sendMessage(
SrvFound, Unicast, 1, id, src )
                                        :: else -> sendMessage( SrvNotFound, Unicast, 0, id,
src ) ;

                                                if
                                                :: msg == 1 -> wait300D = 1
                                                :: else
                                                fi
                                        fi

                                :: else
                                fi
                        fi ;

                        clearMail()
                }

        /*** Announce modes ***/
        :: announceMode == 0 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 3, id, 255 ) ;
                announceMode = 1 ;
                set( waitDelay, 15 ) ;

        :: announceMode == 1 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( IAmCentral, Broadcast, 3, id, 255 ) ;
                announceMode = 2 ;
                set( waitDelay, 15 )

        :: announceMode == 2 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;

                if
```

```
                        :: id == 0 -> sendMessage( SrvRegRqst, Unicast, 0, id, 1 ) ;
                        :: id == 1 -> sendMessage( SrvRegRqst, Unicast, 0, id, 0 ) ;
                        fi ;

                        announceMode = 3 ;
                        set( waitDelay, 30 )

                :: centralNego == 0 && announceMode == 3 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( IAmCentral, Broadcast, 2, id, 255 ) ;
                        set( waitDelay, 30 )

                :: reg300D == 1 && talk300D.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        printf("MSC: UNREG 300D\n") ;
                        reg300D = 0

                :: centralNego == 1 && centralNegoMax.val <= 0 ->
                        atomic
                        {
                                if
                                :: from_mail_to_i ?? [eval(IAmCentral), transType, msg, src, dest] ||
                                        from_mail_to_i ?? [eval(CentralNego), transType, msg, src,
dest] ->
                                                set( centralNegoMax, 30 )
                                :: else ->
                                        centralNego = 0 ;               /* Send announce with 0 first */
                                        announceMode = 3 ;
                                        set( waitDelay, 30 ) ;

                                        sendMessage( IAmCentral, Broadcast, 0, id, 255 ) ;

                                        do
                                        :: my_nego_list ? _, _, _
                                        :: empty( my_nego_list ) -> break
                                        od
                                fi
                        }
                od
        od
}

proctype Searcher( byte id, central )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        timer searchDelay ;
        timer nodeDelay ;
```

```
	do
	:: from_mail_to_i ? msgType, transType, msg, src, dest ->
		delay( nodeDelay, 1 ) ;

		atomic
		{
			if
			:: transType == Broadcast -> skip
			:: transType == Unicast ->
				if
				:: msgType == SrvFound ->
					printf("MSC: FOUND\n") ;
					found300D = 1

				:: msgType == SrvNotFound ->
					printf("MSC: NOT FOUND\n")

				:: else -> delay( nodeDelay, 1 )
				fi
			fi ;

			clearMail()
		}

	:: searchDelay.val <= 0 ->
		delay( nodeDelay, 1 ) ;

		sendMessage( SrvSearch, Unicast, 1, id, central ) ;
		found300D = 0 ;

		set( searchDelay, 30 )
	od
}

init
{
	atomic
	{
		run Node300D( 0, 3, 1 ) ;
		run Node300D( 1, 2, 0 ) ;
		run Searcher( 2, 0 )
	}
}
```

```
// 3-party Consistency Maintenance with message loss
// Ceryen Tan, Vasughi Sundramoorthy
// 8/17 - Message renaming
// 8/16 - Cleanup attempt

#include "dtime.h"

#define N 4
#define MAX_CH 5
#define MAX_LOSS 1
#define Unicast 0
#define Broadcast 1

mtype =
        {
                SubscriptionRqst, SubscriptionRqstAck, SubscriptionRenewal, Resubscribe,
                ServiceUpdate, EventNotification, EventNotifyReq,
                HelloDevice, HelloCentral
        } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte } ;

byte currentService ;
byte numUpdated ;
byte lossCounters[9]

inline getType( msgType )
{
    byte type ;
        if
    :: msgType == SubscriptionRqst -> type = 0
    :: msgType == SubscriptionRqstAck -> type = 1
    :: msgType == SubscriptionRenewal -> type = 2
    :: msgType == Resubscribe -> type = 3
    :: msgType == ServiceUpdate -> type = 4
    :: msgType == HelloCentral -> type = 5
    :: msgType == HelloDevice -> type = 6
    :: msgType == EventNotifyReq -> type = 7
    :: msgType == EventNotification -> type = 8
    :: else -> type = 255
    fi

}

inline sendMessage( msgType, transType, aMsg, aSrc, aDest )
{
    getType( msgType ) ;

    atomic
    {
        if
        :: transType == Unicast ->
            if
            :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
            :: else ->
                if
```

```
                        :: from_mail[aDest] ! msgType, Unicast, aMsg, aSrc, aDest ;
                        :: true -> printf( "MSC: UNICAST: %d, %d, %d\n", aMsg, aSrc, dest ) ;
lossCounters[type]++
                        fi
                    fi ;
                :: transType == Broadcast ->
                    dest = 0 ;

                    do
                    :: dest < N ->
                        if
                        :: dest != id ->
                            if
                            :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                from_mail[dest] ! msgType, transType, aMsg, aSrc, dest
                            :: else ->
                                if
                                :: from_mail[dest] ! msgType, transType, aMsg, aSrc, dest
                                :: true -> printf( "MSC: BROADCAST: %d, %d, %d\n", aMsg, aSrc, dest ) ;
                                    lossCounters[type]++
                                fi
                            fi ;
                        :: else
                        fi ;

                        dest++
                    :: dest >= N -> dest = 0 ; break
                    od ;
                fi ;

                type = 0
        }
}

proctype Node300D( byte id, mode, recepient )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte src ;
        byte dest ;

        /* Central flags */
        byte waitUpdate ;
        byte waitHello ;
        byte service ;

        bit subscribe1 ;
        bit subscribe2 ;
        timer renewal1 ;
        timer renewal2 ;
        timer talk1 ;
```

```
timer talk2 ;

/* Normal flags */
bit sendSubRqst ;
bit sendRenewal ;

timer updateDelay ;

timer nodeDelay ;
timer waitDelay ;

if
:: mode == 1 ->
        sendMessage( ServiceUpdateRqst, Unicast, 0, id, recepient ) ;
        set( waitDelay, 60 ) ;
        waitUpdate = 1 ;

        do
        :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                delay( nodeDelay, 1 ) ;

                atomic
                {
                        if
                        :: transType == Broadcast -> skip

                        :: transType == Unicast ->
                                if
                                :: msgType == HelloCentral ->
                                        waitHello = 0 ;

                                        if
                                        :: waitUpdate == 2 ->
                                                sendMessage( ServiceUpdateRqst, Unicast,
0, id, recepient ) ;
                                                waitUpdate = 1 ;
                                                set( waitDelay, 60 )
                                        :: else
                                        fi

                                :: msgType == ServiceUpdate ->
                                        service = msg ;
                                        waitUpdate = 1 ;
                                        waitHello = 0 ;
                                        set( waitDelay, 60 ) ;

                                        if
                                        :: subscribe1 == 1 ->
                                                sendMessage( ServiceUpdate, Unicast,
service, id, 1 ) ;
                                                set( talk1, 30 )
                                        :: else
                                        fi ;

                                        if
                                        :: subscribe2 == 1 ->
```

```
                                                sendMessage( ServiceUpdate, Unicast,
service, id, 2 ) ;
                                                set( talk2, 30 )
                                        :: else
                                        fi

                                :: msgType == ServiceUpdateRqst ->
                                        sendMessage( ServiceUpdateRqst, Unicast, 0, id,
recepient )

                                :: msgType == SubscriptionRenewal ->
                                        if
                                        :: src == 1 ->
                                                if
                                                :: subscribe1 == 1 -> set( renewal1, 90 )
                                                :: else -> sendMessage( Resubscribe,
Unicast, 0, id, src )
                                                fi
                                        :: src == 2 ->
                                                if
                                                :: subscribe2 == 1 -> set( renewal2, 90 )
                                                :: else -> sendMessage( Resubscribe,
Unicast, 0, id, src )
                                                fi
                                        fi

                                :: msgType == SubscriptionRqst ->
                                        if
                                        :: src == 1 -> subscribe1 = 1 ; set( renewal1, 90 )
                                        :: src == 2 -> subscribe2 = 1 ; set( renewal2, 90 )
                                        fi ;

                                        sendMessage( SubscriptionRqstAck, Unicast, 0, id,
src ) ;
                                        sendMessage( ServiceUpdateRqst, Unicast, 0, id,
recepient )
                                        waitUpdate = 1 ;
                                        set( waitDelay, 60 )

                                :: else
                                        fi
                        fi ;

                        msgType = 0 ;
                        transType = 0 ;
                        msg = 0 ;
                        src = 0 ;
                        dest = 0
                }

        :: waitUpdate == 1 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( HelloDevice, Unicast, 0, id, recepient ) ;
                waitUpdate = 2 ;
                waitHello = 1 ;
                set( waitDelay, 30 )
```

```
                :: waitHello == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( HelloDevice, Unicast, 0, id, recepient ) ;
                        waitHello = 2 ;
                        set( waitDelay, 30 )

                :: waitHello == 2 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;                 /* Service Down? */
                        waitUpdate = 0 ;
                        waitHello = 0 ;

                :: subscribe1 == 1 && renewal1.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        subscribe1 = 0

                :: subscribe2 == 1 && renewal2.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        subscribe2 = 0
                od

        :: mode == 2 ->
                sendMessage( SubscriptionRqst, Unicast, 0, id, recepient ) ;
                sendSubRqst = 1 ;
                sendRenewal = 0 ;
                waitUpdate = 0 ;
                set( waitDelay, 60 ) ;

                do
                :: from_mail_to_i ? msgType, transType, msg, src, dest ->
                        delay( nodeDelay, 1 ) ;

                        atomic
                        {
                                if
                                :: transType == Broadcast -> skip

                                :: transType == Unicast ->
                                        if
                                        :: msgType == ServiceUpdate ->
                                                printf("MSC: %d\n", msg ) ;

                                                if
                                                :: service != msg && msg == currentService ->
numUpdated++
                                                :: else
                                                fi ;

                                                service = msg ;
                                                waitUpdate = 1 ;
                                                set( updateDelay, 60 ) ;

                                        :: msgType == Resubscribe ->
                                                sendMessage( SubscriptionRqst, Unicast, 0, id,
recepient ) ;
                                                sendSubRqst = 1 ;
```

```
                                        sendRenewal = 0 ;
                                        waitUpdate = 0 ;
                                        reset( updateDelay ) ;
                                        set( waitDelay, 60 )

                                :: msgType == SubscriptionRqstAck ->
                                        sendSubRqst = 0 ;
                                        sendRenewal = 1 ;
                                        waitUpdate = 1 ;
                                        set( updateDelay, 60 ) ;
                                        set( waitDelay, 60 )

                                :: else
                                fi
                        fi ;

                        msgType = 0 ;
                        transType = 0 ;
                        msg = 0 ;
                        src = 0 ;
                        dest = 0
                }

        :: sendSubRqst == 1 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( SubscriptionRqst, Unicast, 0, id, recepient ) ;
                set( waitDelay, 60 )

        :: sendRenewal == 1 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( SubscriptionRenewal, Unicast, 0, id, recepient ) ;
                set( waitDelay, 60 )

        :: waitUpdate == 1 && updateDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( ServiceUpdateRqst, Unicast, 0, id, recepient ) ;
                waitUpdate = 2          /* NOW WHAT? */
        od
    fi
}

proctype Node3D( byte id, leader )
{
    chan from_mail_to_i = from_mail[id] ;
    xr from_mail_to_i ;

    /* MESSAGES */
    mtype msgType ;
    byte transType ;
    byte msg ;
    byte src ;
    byte dest ;

    timer nodeDelay ;
    timer updateDelay ;
```

```
	set( updateDelay, 30 ) ;
	currentService = 1 ;
	numUpdated = 0 ;

	do
	:: from_mail_to_i ? msgType, transType, msg, src, dest ->
		delay( nodeDelay, 1 ) ;

		atomic
		{
			if
			:: transType == Broadcast -> skip

			:: transType == Unicast ->
				if
				:: msgType == ServiceUpdateRqst ->
					sendMessage( ServiceUpdate, Unicast, currentService, id,
leader ) ;
					set( updateDelay, 30 )

				:: msgType == HelloDevice ->
					sendMessage( HelloCentral, Unicast, 0, id, leader ) ;

				:: else
				fi
			fi ;

			msgType = 0 ;
			transType = 0 ;
			msg = 0 ;
			src = 0 ;
			dest = 0
		}

	:: updateDelay.val <= 0 ->
		delay( nodeDelay, 1 ) ;

		if
		:: numUpdated == 2 ->
			if
//			:: currentService == 0 -> currentService = 1 ; numUpdated = 0
			:: currentService == 1 -> currentService = 0 ; numUpdated = 0
			:: else
			fi
		:: else
		fi ;

		sendMessage( ServiceUpdate, Unicast, currentService, id, leader ) ;
		set( updateDelay, 30 ) ;
	od
}

init
{
	atomic
	{
```

```
                run Node300D( 0, 1, 3 ) ;
                run Node300D( 1, 2, 0 ) ;
                run Node300D( 2, 2, 0 ) ;
                run Node3D( 3, 0 )
        }
}


// 2-party Consistency Maintenance with message loss
// Ceryen Tan,  Vasughi Sundramoorthy
#include "dtime.h"

#define N 4
#define MAX_CH 5
#define MAX_LOSS 3

#define Unicast 0
#define Broadcast 1

mtype =
        {
                SrvReg, SrvRegRqst,
                ServiceUpdate, ServiceUpdateRqst,
                HelloDevice, HelloCentral,
                SubscriptionRqst, SubscriptionRqstAck, ServiceDown, SubscriptionRenewal,
Resubscribe,
                SrvSearch, SrvFound, SrvNotFound
        } ;
chan from_mail[N] = [MAX_CH] of { mtype, byte, byte, byte, byte, byte } ;

byte currentService ;
byte numUpdated ;
bit found300D = 1
byte lossCounters[14] ;

inline clearMail()
{
        msgType = 0 ;
        transType = 0 ;
        msg = 0 ;
        src = 0 ;
        dest = 0;
        req = 0
}

inline getType( msgType )
{
    byte type ;

    if
    :: msgType == ServiceUpdate-> type = 0
    :: msgType == ServiceUpdateRqst-> type = 1
    :: msgType == HelloDevice -> type = 2
    :: msgType == HelloCentral -> type = 3
    :: msgType == SrvReg-> type = 4
    :: msgType == SubscriptionRqst-> type = 5
```

```
        :: msgType == SubscriptionRqstAck-> type = 6
        :: msgType == SrvRegRqst-> type = 7
        :: msgType == ServiceDown-> type = 8
        :: msgType == SubscriptionRenewal-> type = 9
        :: msgType == Resubscribe -> type = 10
        :: msgType == SrvFound -> type = 11
        :: msgType == SrvNotFound -> type = 12
        :: msgType == SrvSearch -> type = 13
        :: else -> type = 255
        fi
}

inline sendMessage( msgType, transType, aMsg, aMsg1, aSrc, aDest )
{
        getType( msgType ) ;

        atomic
        {
                if
                :: transType == Unicast ->
                        if
                        :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                from_mail[aDest] ! msgType, Unicast, aMsg, aMsg1, aSrc, aDest ;
                        :: else ->
                                if
                                :: from_mail[aDest] ! msgType, Unicast, aMsg, aMsg1, aSrc, aDest ;
                                :: true -> printf( "MSC: UNICAST: %d, %d, %d, %d, %d\n", type, aMsg, aMsg1, aSrc,
dest ) ; lossCounters[type]++
                                fi
                        fi ;
                :: transType == Broadcast ->
                        dest = 0 ;

                        do
                        :: dest < N ->
                                if
                                :: dest != id ->
                                        if
                                        :: type == 255 || lossCounters[type] >= MAX_LOSS ->
                                                from_mail[dest] ! msgType, transType, aMsg, aMsg1, aSrc, dest
                                        :: else ->
                                                if
                                                :: from_mail[dest] ! msgType, transType, aMsg, aMsg1, aSrc, dest
                                                :: true -> printf( "MSC: BROADCAST: %d, 5d, %d, %d\n", aMsg, aMsg1,
aSrc, dest ) ;
                                                        lossCounters[type]++
                                                fi
                                        fi ;
                                :: else
                                fi ;

                                 dest++
                        :: dest >= N -> dest = 0 ; break
                        od ;
                fi ;
```

```
            type = 0
        }
}


proctype Node300D( byte id, mode, recepient )
{
        chan from_mail_to_i = from_mail[id] ;
        xr from_mail_to_i ;

        /* MESSAGES */
        mtype msgType ;
        byte transType ;
        byte msg ;
        byte req ;
        byte src ;
        byte dest ;

        /* MODE 1 */
        byte waitUpdate ;
        byte waitHello ;
        byte Poll;

        bit subscribe1 ;

        timer renewal1 ;
        timer renewal2 ;

        bit waitAck1 ;
        bit waitAck2 ;

        bit wait300D ;
        timer talk300D ;
        bit reg300D ;
        bit suspendService ;
        byte renew1;
        byte renew;
        bit unexpectedUpdate;

        timer talk1 ;
        timer talk2 ;

        byte service ;

        /* MODE 2 */
        bit sendSubRqst ;
        bit sendRenewal ;

        timer updateDelay ;

        timer nodeDelay ;
        timer waitDelay ;
    timer searchDelay ;

        if
        :: mode == 1 -> /*Central mode*/
```

```
		do
		:: from_mail_to_i ? msgType, transType, msg, req, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: transType == Broadcast -> skip

				:: transType == Unicast ->
					if
					:: msgType == HelloCentral ->
						/* TO DO! Service still available -> Ask to re-
register */
						sendMessage( SrvRegRqst, Unicast, 0, 0, id, 2 ) ;  /*
re-registration for node 1 */
						waitHello = 0 ;

					:: msgType == ServiceDown ->
						Poll = 1;
						suspendService = 0;				/*
deactivate: suspend propagating service information */

					:: msgType == SrvReg ->
						if
						:: src == 2 ->
							printf("MSC: REG 300D\n");
							reg300D = 1 ;
							suspendService = 1; /* activate: service can
be propagated to interested SU */
							if
							:: wait300D == 1 -> sendMessage(
SrvFound, Unicast, 1, 0, id, 1 ) ; wait300D = 0
							:: else
							fi
						:: else
						fi

					:: msgType == SrvSearch ->
						if
						:: req == 0 && reg300D == 1 &&  suspendService  -
> sendMessage( SrvFound, Unicast, 0, 1, id, 1 )
						:: else -> sendMessage( SrvNotFound, Unicast, 0, 0,
id, 1 ) ;

							if
							:: req == 0 -> wait300D = 1
							:: else
							fi
						fi

					:: else
					fi
				fi ;

				clearMail()
```

```
                }

            /* wait for prompt from SU that node is down */
            :: Poll == 1 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( HelloDevice, Unicast, 0, 0, id, 2) ;
                waitHello = 1 ;
                Poll=0;
                set( waitDelay, 30 )

            /*poll SM */
            :: waitHello == 1 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;
                sendMessage( HelloDevice, Unicast, 0, 0, id, 2 ) ;
                waitHello = 2 ;
                set( waitDelay, 30 )

            /*poll again SM*/
            :: waitHello == 2 && waitDelay.val <= 0 ->
                delay( nodeDelay, 1 ) ;                    /* NOW WHAT? -> should purge
data in RL */
                waitUpdate = 0 ;
                waitHello = 0 ;
                printf("MSC: Purged node!") ;
                reg300D = 0;
            od


    :: mode == 2 -> /* Subscriber SU */
            sendMessage( SubscriptionRqst, Unicast, 0, 0, id, 2 ) ;
            sendSubRqst = 1 ;
            sendRenewal = 0 ;
            waitUpdate = 0 ;
            subscribe1 = 0;
            set( waitDelay, 60 ) ;

            do
            :: from_mail_to_i ? msgType, transType, msg, req, src, dest ->
                delay( nodeDelay, 1 ) ;

                atomic
                {
                    if
                    :: transType == Broadcast -> skip

                    :: transType == Unicast ->
                        if
                        :: msgType == SubscriptionRqstAck ->
                            sendSubRqst = 0 ;
                            sendRenewal = 1 ;
                            waitUpdate = 1 ;
                            set( updateDelay, 30 );
                            set( waitDelay, 20 )

                        :: msgType == ServiceUpdate ->
                            printf("MSC: %d\n", msg ) ;
```

```
                                        if
                                        :: service != msg && msg == currentService ->
numUpdated++
                                        :: else
                                        fi ;

                                        subscribe1 = 1;
                                        service = msg ;
                                        sendSubRqst = 0 ;
                                        sendRenewal = 1 ;

                                        if
                                        :: waitUpdate==2 -> printf("MSC: RESPONSE TO
UNEXPECTED UPDATE\n") ;
                                                sendMessage( SubscriptionRenewal,
Unicast, 0, 0, id, 2 );
                                                waitUpdate = 1;
                                                set(updateDelay, 30)
                                        :: else
                                        fi;

                                        waitUpdate = 1 ;
                                        unexpectedUpdate = 1;
                                        set( updateDelay, 30 )

                                :: msgType == Resubscribe ->
                                        sendMessage( SubscriptionRqst, Unicast, 0, 0, id, 2 )
;
                                        sendSubRqst = 1 ;
                                        sendRenewal = 0 ;
                                        waitUpdate = 0 ;
                                        reset( updateDelay )

                                :: msgType == SrvFound ->
                                        printf("MSC: FOUND\n") ;
                                        found300D = 1;
                                        waitUpdate=0;

                                        if      /* check if the msg in SrvFound provides the
correct req and update the service attr */
                                        :: req == 0 && service != msg -> numUpdated++
        /* for verification -> []<>(numUpdated >=1)*/
                                        :: else -> wait300D = 1
                                        fi

                                :: msgType == SrvNotFound ->
                                        printf("MSC: NOT FOUND\n") ;
                                        wait300D = 1;

                                :: else
                                        fi
                        fi ;

                        clearMail()
                }
```

```
                :: sendSubRqst == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( SubscriptionRqst, Unicast, 0, 0, id, 2) ;
                        set( waitDelay, 30 )

                :: sendRenewal == 1 && waitDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( SubscriptionRenewal, Unicast, 0, 0, id, 2 ) ;
                        set( waitDelay, 20 )

                :: waitUpdate == 1 && updateDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        printf("MSC: UPDATE1 NOT RECEIVED\n") ;
                        sendMessage( ServiceUpdateRqst, Unicast, 0, 0, id, 2 ) ;
                        waitUpdate = 2;
                        set(updateDelay, 30)

                :: waitUpdate == 2 && updateDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        subscribe1=0;
                        printf("MSC: UPDATE2 NOT RECEIVED\n") ;
                        Poll=1;
                        sendMessage( ServiceDown, Unicast, 0, 0, id, 0); /* NOW WHAT? -> Send
msg to Central notifying service down */
                        found300D = 0 ;
                        waitUpdate = 3;
                        set( searchDelay, 30 )

                :: Poll==1 && waitUpdate==3 ->
                        printf("MSC: Waiting\n") ;
                        waitUpdate = 4;

                /*:: waitUpdate == 2 && unexpectedUpdate == 1 && subscribe1 == 1 -> //unexpected
update
                        delay (nodeDelay, 1);
                        waitUpdate = 1;
                        printf("MSC: RESPONSE TO UNEXPECTED UPDATE\n") ;
                        sendMessage( SubscriptionRenewal, Unicast, 0, 0, id, 2 );
                        waitUpdate = 1;
                        set(updateDelay, 30)
                */

                :: found300D==0 && waitUpdate==4 && searchDelay.val <= 0 ->
                        delay( nodeDelay, 1 ) ;
                        sendMessage( SrvSearch, Unicast, 0, 0, id, 0 ) ;
                        found300D = 0 ;
                        waitUpdate = 3;
                        set( searchDelay, 30 )
                od

        :: mode == 3 -> /* Service Provider SM */
                set( updateDelay, 30 ) ;
                currentService = 1 ;
                numUpdated = 0 ;
```

```
		do
		:: from_mail_to_i ? msgType, transType, msg, req, src, dest ->
			delay( nodeDelay, 1 ) ;

			atomic
			{
				if
				:: transType == Broadcast -> skip

				:: transType == Unicast ->
					if
					:: msgType == SubscriptionRqst ->
						subscribe1 = 1 ;
						renew=1;
						sendMessage( SubscriptionRqstAck, Unicast, 0, 0, id,
1 ) ;
						sendMessage( ServiceUpdate, Unicast,
currentService, 0, id, 1 );
						waitUpdate = 1 ;
						set( renewal1, 40 )

					:: msgType == SubscriptionRenewal ->
						if
						:: subscribe1 == 1 -> set( renewal1, 40 )
						:: else -> sendMessage( Resubscribe, Unicast, 0, 0,
id, 1 )
						fi

					:: msgType == ServiceUpdateRqst ->
						sendMessage( ServiceUpdate, Unicast,
currentService, 0, id, 1) ;
						set( updateDelay, 30 )

					:: msgType == HelloDevice ->
						sendMessage( HelloCentral, Unicast, 0, 0, id, 0) ;

					:: msgType == SrvRegRqst -> /* Re-registration */
						sendMessage( SrvReg, Unicast, currentService, 0, id,
0 )

					:: else

					fi
				fi ;

				clearMail()
			}

		:: subscribe1==1 && waitUpdate==1 && updateDelay.val <= 0 ->
			delay( nodeDelay, 1 ) ;

			if
			:: numUpdated == 1 ->
				if
				:: currentService == 1 -> currentService = 0; numUpdated = 0;
				:: currentService == 0 -> currentService = 2;
```

```
                                        :: else
                                        fi
                                :: else
                                fi ;

                                sendMessage( ServiceUpdate, Unicast, currentService, 0, id, 1 ) ;
                                set( updateDelay, 40 )

                        :: renew==1 && renewal1.val <= 0  ->
                                delay( nodeDelay, 1);
                                renew=2;
                                printf("MSC: RENEW1 NOT RECEIVED\n") ;
                                set (renewal1, 40)

                        :: renew==2 && renewal1.val <=0 ->
                                delay( nodeDelay, 1);
                                renew=0;
                                printf("MSC: RENEW2 NOT RECEIVED\n") ;
                                subscribe1=0;
                                waitUpdate=0;
                        od
        fi
}

init
{
        atomic
        {
                run Node300D( 0, 1, 0 ) ;
                run Node300D( 1, 2, 0 ) ;
                run Node300D( 2, 3, 0 ) ;
        }
}
```