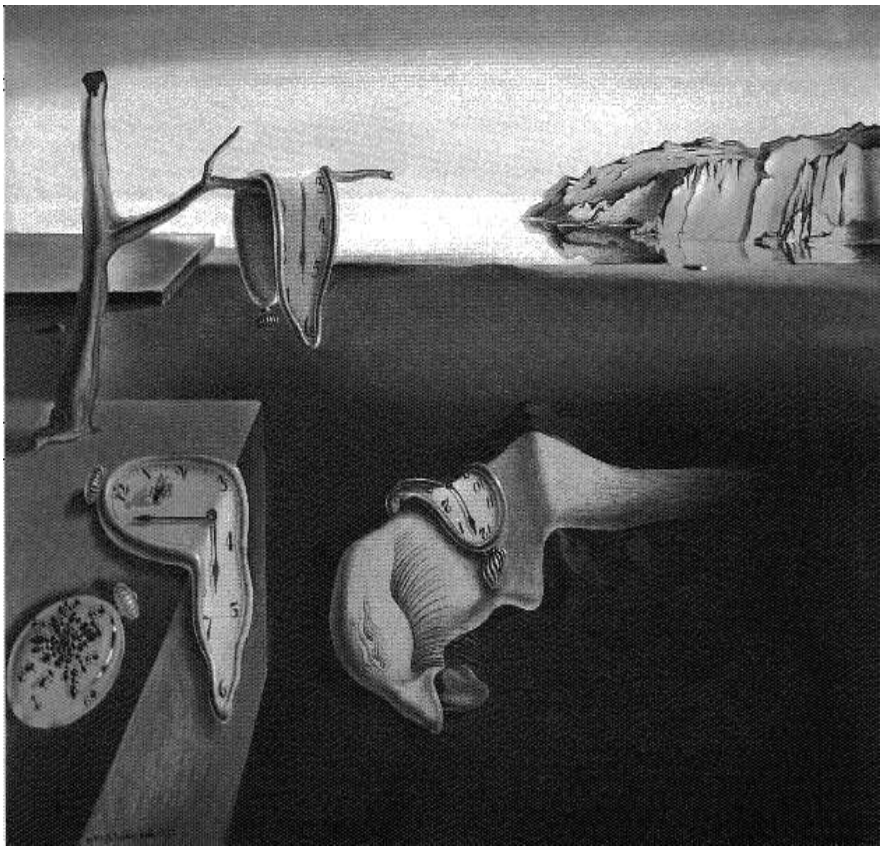
Institut für Automation
Abt. für Automatisierungssysteme

Technische Universität Wien

Projektbericht Nr. 183/1-113
February 2001

Service Specification of W2F

Bettina Weiss



Salvador Dali, "Die Beständigkeit der Erinnerung"

Service Specification of W2F

Bettina Weiss

October 3, 2002

Contents

Abstract

In our project W$_2$F[1] (Wireless/Wireline Factory/Facility Fieldbus), we intend to develop a next-generation LAN/fieldbus which will properly address distribution, security, fault-tolerance, and real-time issues as well as flexibility w.r.t. wireline/wireless interconnections. To achieve these goals, we need a (software) service architecture and a service specification. This report will address these issues.

1 System Description

1.1 Nodes

Every node in the system (sensor, actor, workstation, ...) is connected to the network with a "black box" containing all W$_2$F related hardware, like the network controller, hardware encryption facilities, a secure storage area, ... The box holds all data about the node which is important to the W$_2$F system. This data is downloaded into the box when the node is added to the system.

Basically, nodes can be wireline stations (S) or mobiles (M). Some wireline stations also have wireless transceivers and can be used as base stations (B) to provide access points for the mobiles into the wireline network. Some nodes are small without much computation power (sensors or actors), others are large workstations with lots of storage capacity and processing power.

For every node in the system, we have four different types of problems to solve:

Initialisation: How do we get the node into the system? How do we provide it with its initial secrets, certificates, etc.?

Programming: How do we programm the node? If it is a sensor, how do we make it send its value in W$_2$F and how can other nodes receive and use this value?

Service Consummation: How can a node passively consume a given service?

Service Participation: How can a node participate in a service?

Fortunately, it appears that we can solve all these problems with appropriate services (e.g., initialisation is solved by the initial admission service). However, for the time being we do not concern ourselves with the programming problem. We assume that sensors simply have a way of sending their data to the system in the course of their own sensor service, actors react to commands directed to them from members of their actor service, and all other nodes also can send/receive data and process it.

1.2 Communication

We assume that all communication in the system, both wireline and wireless, uses spread-spectrum CDMA on top of UWB [FGSL01]. The method is quite complex from an electrical engineering point of view, but all we need to know for the time being is that it provides

[1] The W$_2$F-project (*http://www.auto.tuwien.ac.at/Projects/W2F/*) has received funding from the Austrian START-programme Y41-MAT.

a theoretically unlimited number of parallel channels which are each accessed by a unique channel key. A message sent on a channel can be received by anyone possessing the appropriate key. A node can send to at most one channel at a time and can receive from at most k channels at a time. In real systems, the number of different channel keys that are actually used is about 300-500, and k ranges in 20-50. The transmission bandwidth is somewhere in between 1MHz (wireless) and 1GHz (wireline) and the length of the channel key l_{key} is 100-1000 bit.

Since every transmitted bit is encoded with the full key, we get an actual transmission rate of $1/l_{key}$ MBit/s (wireless) and $1/l_{key}$ GBit/s (wireline) per channel.

We call a channel

corrupted if its key is known to more nodes than is allowed in the system,

jammed if it is blocked, i.e., no messages can get through the channel,

good if it is neither corrupted nor jammed.

1.3 Connectivity

Physically, the system consists of interconnected subnets (SN). Subnets are the largest set of fully interconnected nodes, see Figure 1.

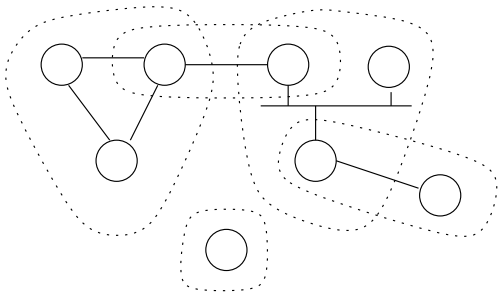


Figure 1: *Subnets*

The subnets are not static, but dynamically change as mobile nodes move between SNs and as communication links are added or fail for a longer period of time. A subnet may consist of only one node if that node does not have a link to any other node in the system (e.g., due to a transceiver fault). SNs are interconnected by nodes which are part of both SNs.

We expect that there is an upper limit on the number of nodes within one SN, but there is no limit on the number of SNs in the system.

We might want to allow subnets to have up to l_c constantly failed links (or maybe every node may have up to l_c failed links to the other nodes), so subnets need not be fully connected. However, the higher the degree of connectivity, the better our algorithms will work.

1.4 Organization

Since all nodes within a subnet are directly connected through a broadcast or a point-to-point channel, communication among them is much faster than between nodes in different

SNs. Therefore, the W$_2$F services will use synchronous algorithms within a subnet and asynchronous algorithms for service users in other subnets.

There are two (perhaps three) ways to implement services with the SNs:

1. The SN which can provide the best QoS implements the service. All other nodes are clients.
2. Every SN implements the service. The local services are coordinated between the SNs through the gateway nodes.
3. Every SN with a sufficient number of members (or a sufficient QoS) implements the service. These SNs are again coordinated to form a global service. All other nodes are clients.

Open Issues:

- Should every subnet provide all services?
- If yes, should the "gateway" nodes participate in all subnets?
- If the gateway nodes participate in all subnets, can oscillations occur?

1.5 Adversary

An important issue for a secure system is to reason about the (expected) power of an adversary or faulty node. On the whole, we can assume two types of adversaries, *malicious* and *non-malicious*.

A non-malicious adversary does not make errors on purpose, but may make any kind of unintentional errors. For example, a faulty node may

- generate a byzantine fault (but only unintentional),
- disclose its secrets to other nodes (only due to HW/SW errors),
- use the secrets that other nodes have disclosed (if HW/SW allows).

But no node may

- try to compromise the secrets of other nodes,
- use non-W$_2$F hardware or software.

A malicious adversary, on the other hand, will try to compromise the secrets of other nodes and may use any hardware or software. Its byzantine behavior will be chosen to cause a maximum of harm to the system.

Clearly, a malicious adversary is more powerful and will require the full set of security measures (secure storage, encryption, ...) to combat. However, these measures may be unnecessary in case of a non-malicious adversary, which is basically a normal faulty node without any malign intentions. We feel that for these nodes some hardware-support and perhaps signatures and certificates may be enough to ensure that no harm is done.

In both cases, we will have to identify the effect on our algorithms and services and devise counter-measures if necessary.

2 Service Management

2.1 Service Structure

Currently, we are investigating two approaches: a *hierarchical* structure and a *distributed* structure. In the rest of the paper, we will assume a distributed structure.

2.1.1 Hierarchical

In the hierarchical structure, a set of trusted nodes forms a *trusted computing base* (TCB) which can initiate the formation of a trusted server base (TSB). This base can in turn form subsequent service groups. If something bad happens to such a group (like a jamming node or an intruder), then the group reports this to its parent, who may take measures like excluding a faulty node or even dissolving the group. A parent group would have to create the group, regularly check the performance of the group (in case the group becomes faulty without noticing it), and destroy the group if necessary. In the last case, the parent would also have to be responsible for notifying all clients of the group that the old group has dissolved and a new group has been formed. If access to the new group is the same as to the old group, then we do not need to inform clients. The advantage of a hierarchical structure is that there is some higher authority to turn to. On the other hand, the TCB is a bottleneck. And of course it must be trusted.

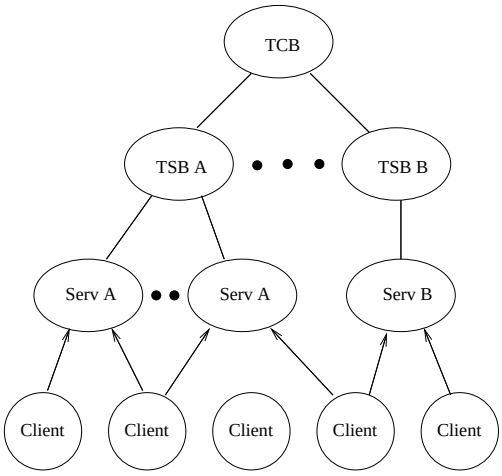


Figure 2: *Hierarchical Service Structure*

Figure 2 shows the service structure. We have one TCB which creates a TSB for every desired service. The TSBs in turn create one or more subservices, which can be used by the clients.

In order to create a group, the parent group must know several things about prospective group members:

- Which nodes are willing to participate?
- Are the nodes able to provide the service? To decide this, the parent must know about a node's

- capabilities
- connectivity
- current load
- trustworthiness

2.1.2 Distributed

In the distributed structure, there is no parent to whom a group or a client may turn in case of problems. The nodes will have to fight out everything among themselves. It has the advantage that we do not need to trust anybody. However, not every problem may have a satisfactory solution in a distributed structure, as we will see in Section 2.8.

2.2 Quality of Service

Before we start to develop our general service structure, we should state what the user requires from services. Generally speaking, we can identify three properties which define the quality of a service (QoS):

- response time
- correctness
- accuracy

Obviously, all services should strive to respond to user requests as fast as possible, and their answers should always be correct. Some services, like clock synchronization, will not have a single correct value, but will reply with an interval which contains the correct value. For such services, the aim should be to make the interval as small as possible while still keeping it correct.

We would like to describe the quality of our services with a function Q which captures the response time and the "worth" of the reply when it is received. The function should result in a small value for low QoS and in a large value for good QoS. However, since quality gets lower if the path from the service provider to the sender gets longer, we will instead use a function for specifying the "non-quality" of a service $\bar{Q} = Q^{-1}$. For clock synchronization, e.g., we might define a rough estimation of the non-quality to be $\bar{Q} = |I| + t_R + 2 * t_R * \rho$, where $|I|$ is the length of the interval, t_R is the response time, and ρ is the clock drift of the user. Optimal quality would be achieved for $\bar{Q} = 0$ (and thus $Q = \infty$): the response has been received at once and it was a single correct value.

With this measure we now have a tool to decide when a subnet should implement a service. Assume that we have some client node p_c which wants to consume service S. For every node p_s which can provide the service, we compute the inverse quality $\bar{Q}^S_{cs} = \bar{Q}^S_s + \bar{Q}^S_{sc}$, where $\bar{Q}^S_s$ is the local non-quality of service on node p_s and $\bar{Q}^S_{sc}$ is the quality loss of the data on its way from p_s to p_c. We will have to create a new service group if $min\{\bar{Q}^S_{sc}\} > \bar{Q}^S_{min}$, that is, if the quality of service exceeds some given bound. The new service group must be created in a subnet near enough to p_c to provide adequate QoS.

2.3 Certificates

Each node receives a set of certificates when it is first added into the W$_2$F system. Among them are certificates stating that a node can provide given services. For every service the node provides, it has one certificate linking the node ID with the service ID. This is the initial service certificate (ISC) which simply states that the human user who has added the node to the system thinks that the node should be allowed to perform the service.

The system itself is responsible for assessing a new node's capabilities and for including it into a service. So whenever a node is taken into a service group, it gets an acting service certificate (ASC) stating that it is now part of the service. The certificate contains the node ID, the service ID, and some sequence number. This certificate may also be the ticket to consumate other services. Of course, passive usage does not require a certificate for consumation, only active usage (like data requests) does require it.

If a node exhibits faults, then that node is removed from the service group and its ASC is permanently revoked. Its ISC, however, is only temporarily revoked to keep a faulty node from re-entering the service at once. So when the node recovers, its ASC is gone, but after a timeout its ISC is valid again and it can re-apply for the group.

Note that depending on the implementation, temporary revocations might require synchronized clocks. In any case, it will require at least local clocks. See the description of the certificate management service in Section 3.5 for details.

2.4 Attacks

What kind of attacks on services can we expect? The target of the attack is to make the service either unavailable or at least to reduce its quality. So an attacker has the following options:

- Conduct a denial-of-service attack. This is done by making so many consecutive requests that either the server or the communication channel is overloaded. In any case, the effect is to degrade the service response time for other nodes.
- Intrude into the service and try to make the service either incorrect or at least less accurate.
- Fake the service, i.e., accept and reply to service requests without proper legitimation.

We will have to develop counter-measures against all of these attacks. A denial-of-service attack on server performance can be countered if the server continuously monitors its load. If it notices that it is being overloaded, it should start to schedule requests, taking care to give a fair share of its resources to all clients. So if a server notices that one or more nodes send significantly more requests than the other nodes, it will only service a given minimum of their requests and will service the surplus requests only if there are no requests from other nodes. A load distribution algorithm might be employed to spread the surplus load among the group. In addition, the server might inquire from the error log service and/or the certification service if the node is marked as faulty. If it is, then the server stops responding to its requests (at least temporarily).

Assume we encounter a denial-of-service attack on the channel, i.e., one or more nodes send so many messages on the channel that other nodes never get the channel at all. First of all, we have to make sure that this can never happen under normal circumstances, so we

assume that the channels do not get overloaded during normal operation. This is ensured by the channel access service. If we assume that under normal conditions every node can send its message within some time, then we can recognize nodes which overload the channel with their messages. These nodes are reported as faulty, notified that they are temporarily removed from the system, and the channel is switched. The faulty nodes are not notified of the switch.

To avoid that a faulty node corrupts the service from within, we will have to make sure that faulty nodes are detected at once and removed from the service. We also have to make sure that there can never be more faulty nodes within the service than the algorithm can handle.

Faking the service is prevented by the service certificates. The only way a node can fake a service is when it has started a service group containing only faulty nodes, when there is no good service group (because the good group would try to merge), and when the clients do not notice that the service is bad. It can happen, but it is very unlikely.

2.5 Initialization

When a new node is added, it must be equipped with some data in order to participate in the system (see also Section 3.5). If we assume that a particular W_2F system is identified by some unique system identifier SID, then the node will have at least

- a unique identifier (UID),
- a public/private key pair K, K^{-1},
- a certificate Cert(UID, SID, K) binding the UID, the SID and the public key K together,
- a set of admission channels on which the node can contact the system.

The UID is necessary to be able to identify a node. The public/private key pair is required for signatures. The private key K^{-1} must be kept in secure storage and should never be disclosed to anyone else. The certificate is issued by some higher authority and ensures that the node UID is allowed in the system SID and has the public key K. The admission channels are the node's entry point to the system.

When a new node is powered up, it selects a free admission channel and begins to send join requests. Eventually, these requests are picked up by the admission service, which will reply and issue the node a private channel. It will also notify the locationing service and the topology service to determine the node's physical and topological location. After the location has been computed, the node can contact the admission service again and request admission to the services it will need.

2.6 Security Requirements

In order to implement security algorithms on the system, we basically need two things:

1. A common communication channel (pairwise or shared among several nodes).
2. Every node must have a secret.

The first item is simple in normal systems, but hard in a SS-CDMA based system because all users of a communication channel require a common key. We need to find out how the users of a channel get this key. Secondly, we need to find a way to protect the users from jamming of the channel.

The second item is never trivial. The question of how to provide a node with a secret is seldomly addressed. We have to find out who should generate the secret and how it gets to the node. We will also need a way to protect the node's secret from unauthorized access. For systems with the non-malicious adversary assumption, it will probably be enough to keep the secret in non-volatile storage and to ensure that the software does not disclose the secret.

If (1) and (2) are both available, then we can use practically any security algorithm. So we have to strive to provide these two items, and if we want to enhance existing security algorithms, then we will have to find ways to incorporate some of the things high-level security algorithms provide into these basic items.

2.7 Service Group Management

Services are generally provided by a group of nodes (for fault-tolerance reasons). Our system should be able to optimize service groups so that the load on the nodes is balanced (otherwise, a very good server might join every group, get overloaded, and in the end reduce the quality of the service). Fault-tolerant and relyable group management is very important in W_2F, it is one of the core problems we have to solve. What should our (distributed) group management algorithm be able to do?

- Create a group.
- Keep track of the number of members.
- Detect faulty members.
- Join a new member.
- Remove an existing member.
- Detect replication of the group (after a network partitioning has been removed).
- Merge a group.

After starting the system, service groups begin to form. When a node is started up, it sends its service certificates to all service groups it can participate in and offers its own services. Three reactions are possible:

- The group does not exist. The node will experience a timeout and assume that it has to form the service group. It will enter itself into the group and broadcast a join request for the service to all nodes to get new members.
- The group exists, the participation offer is denied. The node is not part of the service. It should nevertheless poll the service periodically to avoid that the service is lost (due to some error or because of network partitioning).
- The group exists, the offer is accepted. The node is now a member of the group.

When a group determines that it has too few good members, then it will broadcast a join request. Nodes capable of participating will reply to the request and the group will elect the new member.

When a group determines that it has too many members, it will elect one node and send it a leave request. The node then stops participating in the service. Note that every removal of a node from a service makes the service channels corrupted.

When a group determines that it has a faulty member, it will send that node a leave request and revoke its service certificate.

If two groups providing the same service have formed (due to network partitioning) and the groups detect each other, then the groups will merge into one large group and this new group will remove any surplus members.

We will need an algorithm to determine which node is the best candidate for joining a service. Every candidate will be evaluated (also against the nodes already in the group) and will be selected if it is the best candidate. The criteria for a candidate are its connectivity —a group of well connected servers can provide better (synchronous) services— and its capabilities. The new server might replace an existing node if the existing node provides worse service. In any case, the certificate service is checked to see whether the new node is currently allowed to join the service. The action of the group depends on the reply from the certificate service:

- Timeout: the node is accepted if the group urgently needs to take on new members and if it can afford to accept a potentially faulty member.
- The node's ISC is valid: accept the node.
- The node's ISC is not valid: do not accept the node.

All group action (join, leave) should be logged. Suspicious actions can at least be detected by a human user, or perhaps by some automated tool (perhaps even an artificial intelligence program?).

2.8 Core and Shield Services

For vital services, it may be prudent to split the service group into a group of nodes providing the core service, and a group of nodes providing a shield service to protect the core from direct communication with the user nodes, see Figure 3.

The idea is that the core is a closed service group consisting of trusted nodes which establish a high-quality service within the group. The group provides this service to the shield and the users, but neither shield nor users know how to participate in the core service. The shield nodes also establish the service, using as input both the data of the core nodes and the data of the other shield nodes. Shield nodes always react directly to service requests from users. Core nodes, however, only periodically react to requests, and only if the shield fails to provide the service.

If the service is not vital, then we can allow that either the shield is empty and the core itself provides the service (in this case, service requests are processed periodically) or vice versa (requests are processed at once, but the service nodes may get overloaded).

To allow the core to assess the quality of the shield nodes, the core nodes listen in on the exchanges of the shield service, but consider the nodes in the shield as untrustworthy and

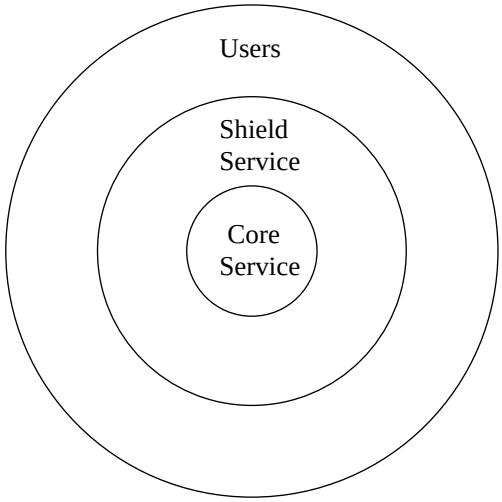


Figure 3: *Core and Shield Services*

do not incorporate their data into the core service. However, they track the performance of the shield nodes because new members of the core service are recruited from the shield.

The communication channels between the three groups core, shield, and users are depicted in Figure 4.

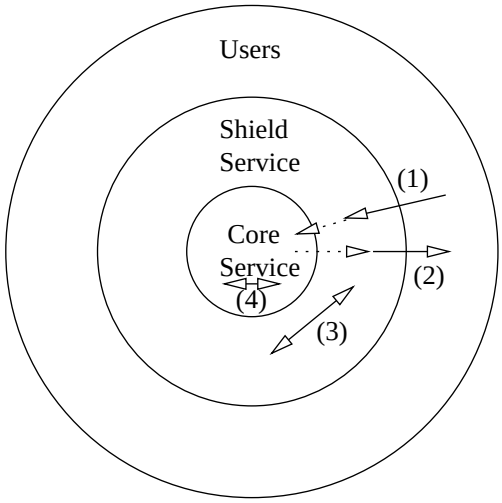


Figure 4: *Interaction between Core, Shield, and Users*

The whole service uses several channels. We assume for the time being that the system (more specifically, the channel switching service) can guarantee that the channels are good (i.e., uncorrupted and not jammed).

(1) The service output channel. It is used by the service to distribute the service data. The channel is common knowledge among all nodes (or at least to the providers and users of the service, which amounts to the same). There is a set of available output channels which is large enough so that a jamming node cannot block all of them at once.

(2) The service input channel. It is used by the user nodes to request the service and is also common knowledge. The shield nodes react to the service requests at once, the core nodes only periodically. There is a set of available input channels which is large enough so that a jamming node cannot block all of them at once.

The input channel can also be used for new nodes to offer their service to the shield.

(3) The shield service channel. This channel is known to the shield nodes and the core nodes. It is used by the shield nodes to establish the shield service, and by the core nodes to monitor the shield service.

(4) The core service channel. This channel is only known to the core nodes. It is used by the core nodes to establish the core service.

The joining/removing of members in the core and shield is shown in Fig 5. We can identify five possible movement scenarios:

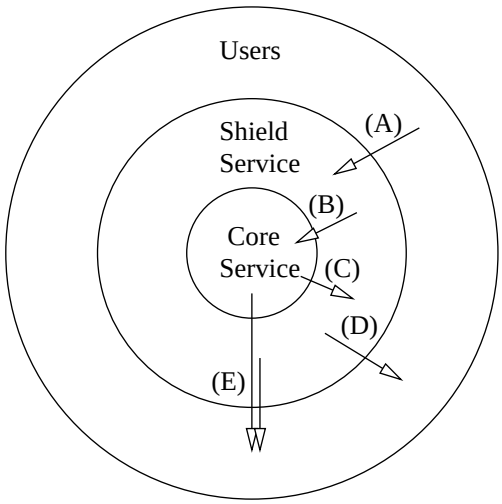


Figure 5: *Member movement in Core and Shield*

(A) **User → Shield**
This happens whenever a user node is accepted within the shield service. It has the following prerequisites:

(a) The user node has a valid ISC.
(b) The shield needs/accepts new members.

(B) **Shield → Core**
This happens when the core service needs new members. The core elects a shield node and contacts this node with the participation offer.

(a) The core needs a new member.
(b) The node has spent some time within the shield.
(c) The quality of service of the node is acceptable for the core service.

(C) **Core → Shield**
This happens when the core has too many members.

(a) The core has too many members.
(b) The node has the worst quality of service of all core nodes.

(D) **Shield → User**
This happens when the shield has too many members.

(a) The shield has too many members.
(b) The node has the worst quality of service of all shield nodes.

(E) **Core/Shield → User**
This happens when the node is faulty. The node's certificate is revoked and the revocation is logged.

(a) The service has agreed that the node is faulty.

When a service is created anew, then the first node which proposes the service is automatically a core node. It will establish the core and shield channels and select the input and output channels from the set of available ones.

Let us assume for the moment that this node is non-faulty. If another node provides its service, the core node elects it as a shield node and inform it of the shield channel. Thus, the core node creates a shield around itself which will maintain itself as soon as it consists of at least one node. Of course, the core monitors the shield and will remove faulty nodes if it encounters them. When a node has been in the shield long enough to be considered trustworthy, the core node will offer it a participation request into the core service. This is a very delicate action as long as the core consists of only a few members. We must ensure at all costs that it can never happen that collaborating faulty nodes can revoke the certificate of a non-faulty node. Therefore, the core node might wait until an appropriate number of shield nodes qualify as core nodes and take them into the core all at once. With this method, we might be able to ensure that there are always enough non-faulty nodes in the core to ensure that faulty nodes do not gain a majority.

What happens if the core node is faulty? Obviously, it might take any number of collaborators directly into the core and might use collaborators as shield nodes as well. We cannot really protect ourselves against the case where a service is only provided by bad nodes or at least by a majority of bad nodes. However, we can at least detect the collaboration if we allow every user node to report a suspected bad service, and if we allow nodes which had their certificates revoked to complain about it. In this case, the human system administrator will at least see the reports and complaints and can manually check the service. Of course, we will have to make sure that the error log service does not get flooded by faked complaints.

The main advantage of discerning between core and shield nodes is that we have only trusted nodes in the core and a set of potentially faulty nodes in the shield. An unknown node may provide its service more or less at once within the shield. But even if the node exhibits faults, it does not corrupt the core service. If the fault protection mechanisms of the shield fail, then the core can still keep up the service, remove the offending node from the shield service, and help the shield nodes to recover.

Can we gain these advantages without requiring such a complex structure and so many nodes? Obviously, with just one service, we will have problems assessing the quality of service

of new nodes without letting them into the service. On the other hand, if we let unknown nodes join the service, then we might add faulty nodes and threaten the correctness of the service.

A possible solution to this dilemma is that every node in the service internally splits the service group into a set of trusted nodes whose data is always accepted, and into a set of non-trusted nodes whose data is only accepted if it conforms to that of the trusted nodes. So for every node, the trusted nodes form the core, and the untrusted nodes form the shield.

Of course, the feasibility of the approach strongly depends on the service in question. For clock synchronization, e.g., it is easy to check the input of untrusted nodes against that of the trusted nodes. There may be services where it is not so easy to decide whether a shield node provides adequate data.

The second advantage of the core/shield concept is that core nodes cannot get overloaded. But as we have already explained, overloading can be seen as a denial-of-service attack and can be combated by reducing the rate with which service requests are processed.

3 Services

3.1 Communication

The common concept in wireline and wireline communication is the "neighbour", i.e., a node which can be directly reached. In a wireline system all nodes on the bus are neighbours; in a wireless system all nodes within communication reach are neighbours. Typically, wireline neighbours will remain longer than wireless neighbours, but this strongly depends on the environment. We can assume that in both systems neighbours may change (be removed or added) without further notice. So our algorithms should not depend too strongly on the network topology.

When we look at the basic communication, we can see that at first we need a communication service to communicate with a neighbour. This is achieved by the *wireless* resp. *wireline datagram service* executed on the mobile resp. station. Base stations can do both. These services are used to establish the current neighbours for all nodes in the system, i.e., to establish the current network topology. On top of it, *routing* can be implemented to create a (global) *datagram service* to allow communication between any two nodes in the system. Of course, due to the dynamic system changes, the network topology must be updated constantly and routing tables must be changed accordingly.

The *local channel service* is responsible for selecting a sender channel and the current set of receiver channels. It is controlled by higher services which tell the local channel service which channels to set up.

The *channel switching service* selects channels for a group which are not compromised and not jammed. The service should be transparent to the higher services. However, higher services can inform the channel switching service that a channel has been corrupted (jamming is detected by the channel switching service itself). To be able to switch a shared channel, the service must know which nodes are currently using the channel (group members and clients) and who should know about the new channel. If the channel is not jammed, then it can be used to transmit the new channel to all valid nodes, using encryption to ensure

that excluded nodes do not know the new channel. If the channel is jammed, then there must be an algorithm that can locally determine the new channel to which the group should switch. This algorithm must yield the same result on all valid nodes, but must not be reproduceable by the jamming node. Currently, we do not know how to achieve this. It is possible that jamming cannot be combated without either physically removing the jamming node or dissolving the service group.

The *channel access service* is responsible for managing fair access to a channel. It should make sure that the channel is not overloaded and that every node gains access to the channel within some given time.

We might also consider some *channel key service* which selects channels for services. The service can store the full set of possible channel keys (recall that not every potential key is a good CDMA key) and select one channel for use by the channel switch service.

3.2 Topology

The foremost issue in the topology management is the *network topology service*, i.e., the determination of direct links and indirect paths between nodes. This is necessary for routing or for the recognition of network splitting.

A second issue is the *locationing service*, which assigns each node a set of 3D-coordinates relative to some common coordinate system. We believe that through transmission delay measurements and comparisons with the data of other nodes, every node can pinpoint the location of every wireless neighbour. We require at least five nodes to pinpoint the location of a sixth node (three to get the 3D coordinates, and two to reduce the error probability[2]).

A related service, which is strictly speaking part of the security, is the *jammer location service*, which can pinpoint the 3D location of a jamming node. It relies upon the locationing service.

3.3 Time Management

We will need a *local time service*, which is a very simple local service that keeps track of the status of the local clock. It manages the clock's parameters like the rate, and can be used to measure relative durations.

The *clock synchronization service* keeps track of the global time. The service is used by the group members to exchange clock values and to synchronize the rates and states of their clocks accordingly. Service users simply synchronize to the group members without sending their own clock value.

3.4 Group Management

The *group management service* has already been outlined. It is responsible for adding resp. removing nodes to resp. from a group. It also keeps track of participation requests and the

[2]Photogrammetric applications also use five points to determine the position of a picture in the coordinate system, and I strongly suspect they have a good reason to require five instead of the four strictly necessary ones.

quality of service of nodes and uses this information to decide which node to add or remove from the group.

The *admission control service* is responsible for allowing nodes to participate in service groups. It is part of the group management service and works hand in hand with the service in question. It selects the node best fit for admission into the group. The admission control service is responsible for collecting and storing participation offers from nodes, for assessing the quality of service of potential candidates, and at the request from the service electing a node from all possible candidates. We expect that admission control will consist of one general part which is the same for all services, and one customized part which is unique for every service. Of course, if we can, we will make the whole service general.

Note that it might be a good idea to devise benchmarks to test the performance of a new node. We could either use general benchmarks (send the node a general task like counting and check how fast it performs the task) or use service-specific benchmarks.

The *load distribution service* is used to distribute the load of a service group among its members. This only applies to services with active users, not to services where users listen passively. The service is useful if a request does not need to be processed by all members of the group but only by a subset of them. In this case, we have to ensure that all members get their fair share of the workload according to their abilities.

The current load on a node is determined by the *local load management service*, which knows the node's maximum allowed load (both processor and service request load) and also manages load bounds set on a per-service basis. If the load manager determines that a service transgresses the load bounds of the node, it notifies the load distribution service to move the excess load to the other servers.

3.5 Security

The *initial admission* of a node into the W$_2$F system is done by a single "admission authority" (AA). This type of admission should not be confused with the admission control service which is responsible for admitting nodes into groups and which will be described later on.

The AA is a dedicated server which allows a human user to add a new component to the system. The idea is that system access is only achieved by customized W$_2$F hardware which has an interface to whatever the component itself uses for communication and an interface to the W$_2$F communication system. The box has several tasks:

- Allow the component access to the W$_2$F network.
- Translate the component communication method to the W$_2$F method. Ideally, it should also be able to translate communication protocols. In this case, the human user could install the original system on top of the W$_2$F communication architecture without any changes, i.e., W$_2$F would provide downward compatibility.
- Store all data required for participation in the W$_2$F system. Keys should be kept in a secure storage area, and in any case all data should probably be kept in non-volatile storage.
- Perhaps provide vital security functions like encryption or computation of shared keys to avoid that the component secret ever leaves the box.

The AA, with the help of the human user, enters the following component data into the box (all previous data in the box is overwritten):

- A unique component identifier UID.
- The type of the component (sensor, actor, server, ...).
- A certificate for the component. It is used as proof that the component has been added to the system by the AA.
- A unique secret only known to the component. It can be used by the component to establish shared encryption keys. If this secret is compromised, the box must be blanked and the component installed anew.
- Optionally a private/public key pair for asymmetric encryption.
- A list of access certificates for the services which the component requires.
- A list of service certificates for the services which the component can provide.
- Basic access information like the channel keys for service access etc.
- Special component data (in case of a sensor, e.g., data ID and data type).
- The software of the component.

In order to ensure that no compromised box is added (e.g., a box which has been constructed by an intruder), the box must be shipped with a certificate ensuring that this box has been constructed by an authorized company. The AA only accepts boxes which contain this certificate, and it must be ensured that the certificate is not readable to anyone except the AA.

Note that the AA must keep all its data (especially the first unused node-ID) in non-volatile storage. It should also keep its method for computing the component secret in secure storage. The secret should probably be computed from a master secret of the AA (which is perhaps kept in the AA's own W_2F box) and the node identifier using a one-way function.

The AA is only required for adding new nodes to the system and is never used by the system itself. So it does not need to be available at all times and we do not require a replicated service to ensure high availability.

Note that the same mechanism can be used to add new services to the system. The service simply gets a unique ID and is added into the AA's list of system services. From then on, new nodes can get certificates for the new service. But we will have to provide a mechanism to equip old nodes with the certificate of a new service without having to take them out of the system to re-initialize their box contents.

Authentication is a pure security service which is responsible for ensuring that a node cannot take on the identity of some other node. The problem can be solved by assigning each node an uncompromised private/public key pair and by requiring each node to sign every message it sends. The service might be incorporated into the W_2F box and might be transparent to the node, signing every message the node sends and filtering out incorrectly signed incoming messages.

The *key distribution service* is used to establish shared keys between two or more nodes. If we use only asymmetric encryption, then we may not need this service at all. However, shared keys are beneficial to security because they get changed more often and more easily

than public keys and thus their compromisation does not carry such a high penalty. We might even consider the CDMA channel keys to be such shared keys, which are used for a short time before being exchanged for new keys. Compromising such a key would have only a short effect before the channel is switched. In this case, we need a function to securely compute a valid new channel key from the secrets of the channel users[3].

The *fault recognition service* is built upon the channel switching service and is responsible for detecting both channel and node faults. Channel jamming is reported by the channel switching service, channel compromisation by the group management service. Node faults are either reported by other security services or by the group management service.

The fault recognition service is responsible for reporting the error to the error log, and in case that the group management service resp. the channel service is not aware of the node resp. channel fault it should also notify the appropriate service. In case of a node fault it alerts the group management service to remove the node from the group and to revoke its certificate, and in case of a channel fault the channel switching service is prompted to change the channel.

The *accounting service* and the *error log service* are both used to keep track of the system state. Accounting notes every change in the system. Whenever a node is added to a group or removed from a group, the accounting service is notified of the change. The error log, on the other hand, gets notified by the fault recognition service whenever a system fault is detected. This could be a jammed channel or a faulty node. In case of a faulty node, the error log might use the locationing service to pinpoint the location of the faulty node. In case of a jammed channel, it might use the jammer location service to pinpoint the location of the jammer.

The *certificate managing service* is used to keep track of revoked certificates. Certificates are only checked during group state changes, when a node is considered for entry into a group, so the service is not used all the time. The service might be part of the error log service because it should be notified whenever a service group determines that a member is faulty and revokes its certificate (or requests certificate revokation from the certificate managing service). Note that the service does not issue new certificates, it only revokes existing ones and only with the consent of the appropriate service group. Only the initial admission server can issue new certificates.

If we use public key encryption or if we use secrets which can be compromised and should be replaced online without reinstalling the node, then we will need a *Certification Authority* (CA). The CA is used to create new secrets and new public/private key pairs to replace compromised ones. In case of asymmetric encryption, it also keeps the revocation list and any node can ask the CA for the current public key of another node. We may need the CA anyway to keep track of certificate revocation.

Keep in mind that a CA for managing keys generally is bad news. Most of the time, things are a lot easier if nodes can handle everything by themselves without asking some higher authority for help. The CA introduces the need for additional protocols (all of which assume that the CA is never under any circumstances compromised) and communication

[3]Unfortunately, not all channel keys are available for use because some keys are better suited for the needs of CDMA than others.

with it is time-consuming. The quorum method employed in COCA [ZSvR00] might provide a partial solution because it allows the CA to be imperfect.

3.6 Service Dependencies

Figure 6 shows the service dependecies. The figure is preliminary and may not include all services mentioned in the sections above.

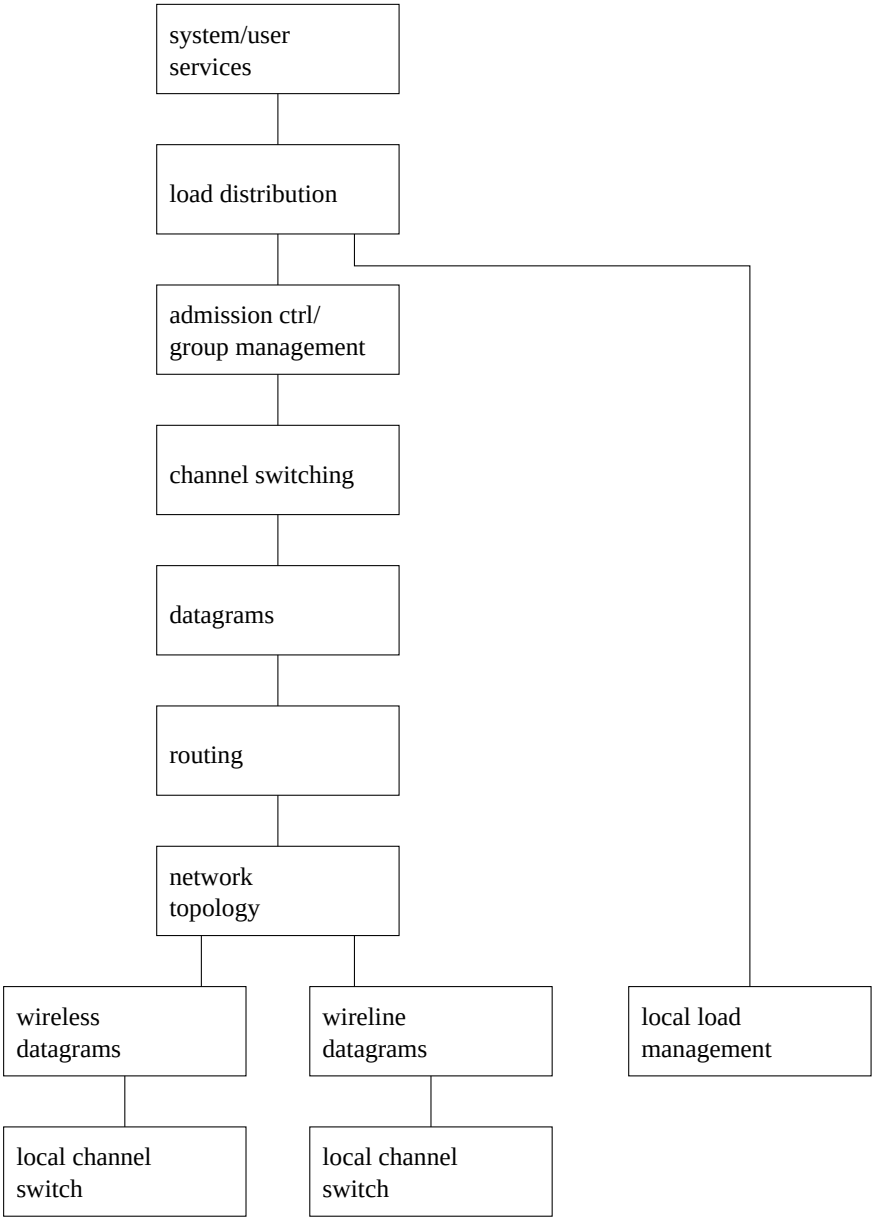


Figure 6: *Service Dependencies (Draft!)*

4 Conclusions

[The text below is what we aim to do, not what has currently been done...]

We have presented an overview on the service structure of our W2F project. We have identified the service requirements and described how we intend to solve them.

References

[FGSL01] Jeff Foerster, Evan Green, Srinivasa Somayazulu, and David Leeper. Ultra-wideband technology for short- or medium-range wireless communication. *Intel Technology Journal*, Q2:(11 pages), 2001.

[ZSvR00] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. COCA: A secure distributed on-line certification authority. Technical Report TR2000-1828, Computer Science Department, Cornell University, December 2000.