S38.3115 Signaling Protocols – Lecture Notes

Lecture 16 – Summary of Signaling System Design

Contents

Introduction

In this lesson we summarize our discussion of signaling system development using our models of the requirements space and design space. Our goal has been to identify invariants – things that have lasting importance over and above aspects that change over time as we move from older systems to more modern ones. For signaling we have said that the driving invariants are: (a) people's need to communicate and (b) the need of the network and service provider's to make a living by earning money from services provision. These invariants translate into *requirements* onto the production machinery of networks that is used to provide communications services and into ways of meeting those requirements – i.e. *designs*. The design encompasses the breakdown of the overall functionality into functional entities and the selection of protocol mechanisms that are used in those entities to meet the requirements. So, actually one

more invariant emerges (c): the methodology of design (that changes only very gradually).

Throughout the course we have used the model of the requirements space presented in Figure 16.1.

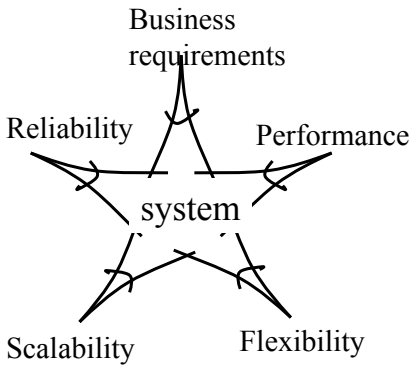

Figure 16.1: Model of the requirements' space

The model focuses on non-functional requirements. All functional requirements: the "what and how" are grouped under the term "Business requirements".

In this course we have also discussed how the requirements of a particular signaling system have been translated to its design and have mind mapped the design space.

In this lesson we will discuss how the requirements of Figure 16.1 have been applicable to different signaling systems, summarize how the importance of the categories of requirements has changed over time and how the requirements relate to each other. We will also discuss the process of translation of the requirements to design.

Moreover, we claim that the requirements space of Figure 16.1 is applicable to many other ICT systems besides signaling protocols. In different fields of applications many of the requirements are expressed slightly differently: different questions may be asked related to, for example, scalability; or performance may be measured differently. But the categories of the requirements stay about the same.

One generic aspect of requirements engineering is that one should have respect for the concept of requirement. There is no room for carelessness on this stage: a mistake at this stage is often very expensive. Even one number or one word in a requirements document of e.g. 10 pages may increase the cost of the resulting system by half or it may render it impossible to implement. So, before one starts translating the requirements to a design, it makes sense to *validate the requirements*. The process of design will give more information on the feasibility of the requirements. This should be fed back to the requirements specification.

There is no single method that could be used to do the validation. However, the goal of the validation is to (a) identify and modify or remove requirements that are impossible to implement, (b) identify requirements that are contradictory to other requirements and resolve the conflicts. Often this means finding a balance between the conflicting requirements. Some times it means choosing the right set of requirements. For the more technical requirements (such as reliability and scalability) it is important that they are *traceable* to some business need.

An example of an impossible requirement is 100% reliability. One nevertheless sees claims of 100% reliability of sold systems – but this is just sales talk and not to be

taken seriously. The cost of reaching a certain level of reliability tends to be governed by the following considerations:

- A certain technology (e.g. a PC or a router) has a base level reliability that is relatively easy to achieve given the available component base. Let this reliability be for example 0.9995.
- For this example getting to 0.9999995 increases the cost significantly (up-to doubling the cost). Hitting any reliability figure between those two may not be easily feasible. Rather the choice is between some discrete steps that relate to duplicating or adding some redundancy schema to particular parts of the whole system.
- If a single weakest link can be identified, less than duplication of cost may fix the problem and let the designer reach some intermediate level of reliability between the two figures above.

Another example of an impossible requirement is when a performance mark that requires exceeding the speed of light in fiber is set. These things happen for example when one asks the users (non-engineers) to set the requirements. Rather than expecting to see such a requirement written down explicitly, one should analyze the implications of performance requirements expressed in some form closer to user perception.

Figure 16.2 shows how the translation is actually a process where the designer may have to go back and forth between the requirements and the design spaces. The reason is that before one can finalize the requirements, one needs to make sure that they are feasible, i.e. it is possible to implement them.

In the design process, one has to break down the overall functionality into functional entities, set the requirements for each of the entities, define the interactions between the entities and select the mechanisms that are used in each entity. In addition, one needs to nail down a number of details related to the mechanisms.

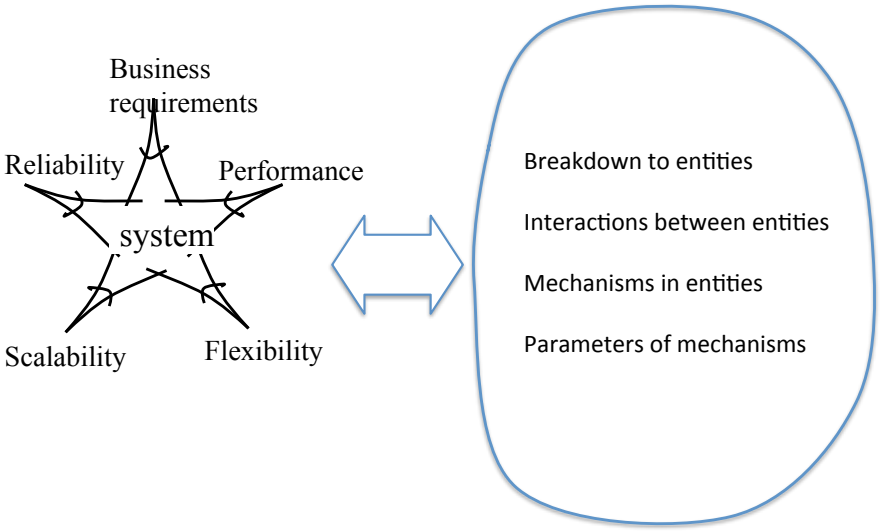

Figure 16.2 Translation of requirements to designs

Why does the model of the requirements space look like the one in Figure 16.1? A simple answer is that this is based on experience. Each category of requirements is as

different as possible from the others. As it turns out, however, they are not truly orthogonal, i.e. independent of each other. We will discuss the relations of the categories of requirements with each other. Each category has its own typical methods of modeling and distinct ways for taking the requirements into account in the design.

Standardization bodies such as ITU-T often mandate the use of the so-called *functional method of specification*. This means that in standardization one first focuses on the functional entities, on information flows between the entities, then translating the flows into protocols and leave the grouping of the entities to products to vendors and the operators. Rarely, the standards even mention the ways, how operators are going to use the technology to earn money. Therefore, a new learner of a signaling protocol typically just has to deduce a lot of the really important stuff that relates to the business aspects. In the approach, we are using on this course, a lot of the specification work under "functional method of specification" falls under design.

The process of development of a new signaling system is recursive in nature: one starts with requirements, moves to design, once after iteration both the requirements and the design are agreed upon, the design on the architectural level is taken as the requirements for the next stage of adding details and moving towards an executable implementation.

Because of the iterative nature of specifying the requirements and designing of a signaling system, in the next sections we discuss both the requirement and design aspects of each of the categories of requirements in Figure 16.1.

Business requirements and design

A methodology that nicely covers the area of business requirements and design is *an adaptation of Open Distributed Processing* (ODP). Figure 16.3 shows the high level structure of the method.

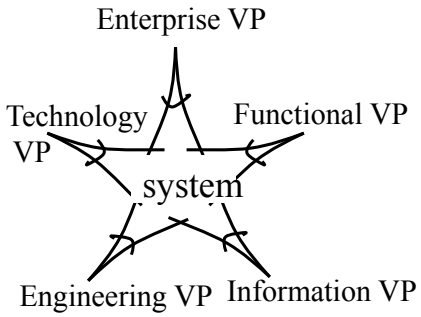

Figure 16.3 Adapted Open Distributed Processing

We define 5 orthogonal viewpoints. They are the *Enterprise viewpoint*, *Functional viewpoint*, *Information viewpoint*, *Engineering viewpoint* and *Technology viewpoint*.[1]

When we apply the method into the development of signaling systems specifications, in terms of our model of Figure 16.1, the enterprise and technology viewpoints fall under Requirements while most of the work under all the other viewpoints falls into design. In each of the viewpoints we can use methods that are typical of that viewpoint.

[1] In ODP "functional vp" is called computational viewpoint. We talk about functional vp, because the result we are after is a specification rather than an implementation.

Enterprise viewpoint

We can use the stakeholder model to specify the stakeholders and their roles in a scenario. Stakeholders are business entities or actors such as different types of operators, subscribers and vendors. Under this viewpoint, the designer should consider things like

- Purpose of the protocol(s)
- Scope of the functionality
- Who can communicate with whom
- Relations of the stakeholders including trust assumptions, customer/supplier or cooperative relations of the actors, how much operators can control what the subscribers can do, etc.
- Main use scenarios.
- Ways of making money with the technology under specification

When defining the stakeholders and their roles, it is important to pay attention to the core competences that are needed to use the technology. When an official standardization body is doing the work, the actors who are present typically hold their cards close to their chest and shy away from any explicit work under enterprise viewpoint with excuses like “this is for the operators or vendors to work out by themselves”. This is because the actors are competitors on the market.

Relations between stakeholders

A recurring issue in the stakeholder model is how much the operators can control what the subscribers are doing? And what freedoms do the subscribers have in modifying the services or creating their own services?

The classical PSTN model of services creation is that it is the operator(s) who provide(s) the service and the subscriber is just a consumer. The Internet model stresses the importance of *user innovation*. The logic is that it is the users themselves who know best what their needs are and no operator can compete in this respect however competent people the operator has. User innovation also requires that experimentation is encouraged and easy to get into.

The second recurring issue is the breakdown of functionality to service provider (SP) and network provider (NP). NP owns a lot of the infrastructure that is expensive to build. There is a strong belief that the users are not interested in the infra but rather in the services. Therefore the money comes to SPs and NPs carry the costs. On the other hand if there are no SPs or their business position is weak, competition is low and the users will not get what they want.

The SP to NP relation is an example of a customer/ supplier relation – a typical phenomenon in a market economy. The tension of the relation comes from the fact that the network/service coverage areas in terms of customer base/geography of both players are often the same.

The third recurring question is how do the operators more generically relate to each other on an operator-to-operator interface? The relation can be a sort of (a) a customer/supplier relation or a (b) cooperative relation. An example of (a) is the case of leasing network capacity by one operator from another or buying transit

connectivity by a local ISP from an international ISP (in this case the coverage areas of the two operators are not the same).

There are many examples of cooperative relations between operators, like (i) routing in the Internet using BGP is cooperative in nature, (ii) peering of ISPs in a peering point for exchange of traffic, (iii) service implementation of many services in circuit switched PSTN/ISDN/GSM networks requires that all operators on the way of the communication support the service. This is because signaling in circuit switched networks is often service dependent (there is service specific information in DSS1, ISUP etc.). A very important example of cooperation between operators is (iv) roaming. This means that subscribers of one operator are allowed to make use of the services of another and the two operators will settle billing/inter-operator accounting with each other without the subscriber having to worry about it (except possibly paying a high tariff).

A relation of operators that is supposed to be cooperative is also subject to competition because competition is in the nature of companies in a market economy. The protocols that are used to implement the cooperation often make it possible to use the other parties' resources to gain business advantage. Sometimes this is called "tussle". An example is the roaming charges: a mobile operator may charge high prices for its services from the roaming customers who then complain to their own operator to whom they pay the bills. At the same time the mobile operators realize that a lot of the value of mobile service lies in the freedom of being connected wherever you are and pricing should not alienate the customers as a whole.

On the course we have noted the trend of signaling and call control architectures on the path ISDN→IN →GSM/CAP → VOIP from cooperative service implementation among many operators towards a clear responsibility of the subscriber's own operator for the service that the subscriber is using. Nevertheless, even in IMS the visited network operator uses its own proxy (P-CSCF) to follow what the foreign subscriber is doing and both operators are able to compare their records of resource use and the trust can be based on that comparison (on application layer).

Adoption and deployment

In order for a protocol or technology to be adopted, the investor into the technology must gain a benefit. I.e. the investment must give revenue that pays back the investment. This seems like a trivial and self-evident rule. However, in the Internet community, people often forget the rule because many aspects of the network are cooperative in nature and cooperation is assumed to imply altruism.

However, the theory of evolution (recall Charles Darwin and e.g. Robert Axelrod) and cooperation shows that cooperation does not necessarily imply altruism. For example the most famous and successful cooperation strategy is tit-for-tat. This strategy says: *do not cheat first but if you are cheated do not turn the other cheek but pay in kind.* Altruism would equal to turning the other cheek i.e. immediately forgetting about the cheating. If tit-for-tat is a fair strategy in the sense that a player does not cheat first, there is a mirror image of the strategy of turning the other cheek, called Win-stay; loose- switch. This strategy allows cheating many times and becomes attractive when among the players there are ones who tend to turn the other cheek. Win-stay, loose-switch becomes dominant under such circumstances.

One should consider the rule of alignment of investments and benefits when allocating the responsibility for certain functionality to one or the other operator in a wide area network. There are numerous examples where the designers of protocols have forgotten the rule (e.g. IPv6) where the result is slow or failed adoption.

Carrier Grade vs. Best Effort

The classical approach to services provisioning by the operators is that services should be carrier grade (CG). A CG service is one where the operator is able to manage permissions at points (A, B, C….) in their network like: at A: carry traffic X to B. This excludes any leakage of traffic from any X to any C ≠ B. If no permission can be found, then no traffic will be carried. This requires that, what happens can be predicted. Both parties know what to expect.

For example, SDH managed connections and PSTN/ISDN/GSM/3G calls are like that. On the other hand classical Ethernet, IP, Best-effort service, IP/MPLS do not satisfy the above definition.

The motivation for CG is that it will make services similar to ordinary goods in a market economy: it requires that the supplier knows what he is selling and takes the responsibility for the quality of the goods supplied. Also, the buyer knows what he is buying and has the consumer rights protected in the transaction.

Leakage of information takes place in MAC learning in Ethernet. IP-routing is cooperative in nature and therefore actions by some remote ISP can have an impact on how traffic is carried in the local ISP network. Identification is poor in IP networks. Therefore spoofing of source addresses and DDoS attacks are possible leading to unpredictable availability. IP/MPLS relies on IP routing. Best-effort means that even one user can hog as much network capacity as e.g. 100 other users – so communication may fail because of what other people or systems are doing etc. These are examples of how communications services and technologies deviate from the idea of carrier grade.

We see that in modern networks both the ideas of carrier grade services and best effort services have significant support. CG requires extensive protocol support in a packet network. A lot of the work in IETF and other standardization bodies and industrial fora relates to achieving CG for some services. A recent example is the idea of Software Defined Networking. One can expect that these two approaches to networks and network services will continue to compete over the next years.

Functional viewpoint

Once the enterprise level requirements are agreed upon, it is time to break down the scope of the functionality into functional entities and define the interactions between the entities. Methods like signaling flow charts and SDL (Specification and description language) can be used. It is typical of signaling protocols that there are many interactions between the entities. We say that signaling protocols are control centric rather than information centric.

On a more detailed level, we select specific mechanisms that are used in each of the entities.

It is important that the functions and the mechanisms can be traced back to either the enterprise level business requirements or to the non-functional requirements (reliability, performance, scalability, flexibility). The traceability helps to make sound decisions when we are updating or modifying the specifications or reusing a known protocol for a new purpose. When an interface between different stakeholders of the enterprise viewpoint maps to a clearly defined protocol, the design will support the assumed way of doing business. Alternatively, the business level design may be of a more "conceptual nature".

Often it is the desire of flexibility that leads to modular designs and this has an impact into the breakdown of the functionality into functional entities. Reliability requirements often map to a selection of particular protocol level mechanisms that are used to reach the required level of reliability.

Information viewpoint

Under information viewpoint we specify the information carried and processed by the protocol. The information can be for example in a binary or textual format.

The first thing to define, based on the business requirement of who can communicate with whom, is how to identify the communicating parties, the sessions and other objects that are needed for the sessions. An example is: use E.164 telephone numbers for identifying callers and callees (or use URIs of some sort) and call references (a'la DSS1) to identify calls. The methods of identification are difficult to change in a protocol that is used on live network, so once fixed, they tend to set strong boundaries for the applicability of the protocol. The difficulty lies in the need to make the changes without disrupting the services to the users.

Historically, an important issue in signaling protocol design has been how to tie the voice circuit with the signaling and call control FSM. In channel associated signaling (CAS) the mapping is one-to-one, i.e. there can be no signaling unless a voice circuit is reserved. Such signaling is not suitable for example to implement mobility. Mobility requires signaling (for location updates, handovers etc.) that is voice circuit independent. Another example where circuit independent signaling is needed is look-ahead. Look-ahead is implemented for example in VOIP where the first thing to find out with signaling is the callee's willingness to communicate (without reserving any resources for voice). Besides CAS, in ISUP the signaling FSM and the voice circuit are linked 1-to-1 by the Channel ID Code (CIC). This makes the strong assumption of hop-by-hop routeing of the call and excludes look-ahead. Another consequence is that for Intelligent Networking a different protocol (INAP) is needed instead of running ISUP directly to the node housing the services logic.

Engineering viewpoint

The designers of the signaling systems often leave this work to the vendors and operators. Under this viewpoint we map the functional entities into equipment. When the equipment is installed into a network, the result is the production infrastructure that will support the signaling protocols and the services that can be created on top. The designers usually try to write the specifications in a way that several different mappings to equipment are possible.

Technology viewpoint

Under technology viewpoint we specify the requirements for the underlying technologies, like over which kinds of networks and protocols one can run the signaling protocol under design. Historically, for example the assumption of PDH transmission with its 64 kbit/s signaling and voice channels has had a strong impact on the signaling protocols that were designed.

In an IP network it may be possible to run the current protocol under design over several different stacks of protocols for example depending on the desired level of security and reliability. It makes sense to decide on different alternative underlying protocols early in the process of specifying a new protocol. This allows limiting the scope of functionality to the truly necessary aspects that are not covered by the underlying protocols.

Performance requirements

Performance is measured in terms of *delay*: how long does it take to execute a function.

We can measure it from the point of view of (1) the end user or (2) the operator's network or (3) a single network node or (4) a single hop over a network.

From the end user's point of view an important criteria for signaling system performance is *post dialing delay*, i.e. the delay for the last digit dialed till the ringing tone. To measure the post dialing delay one needs to assume a network or call model such as local network/call; call over a national network; an international call. The post-dialing delay is a good measure from the point of view of the end user. It is an example how one can measure the *responsiveness of the system to user actions*.

Once the end-to-end delay requirement is set, this becomes a delay budget that can be used in the subsystems (such as network nodes) that are on the end-to-end path. Breaking down an end-to-end delay budget to the subsystems is an element of design under "performance".

To make sound choices of technology an operator may use delay requirements for a node such as, how long does it take for the node to process a particular message in the signaling protocol. For example for processing ISDN protocol messages such requirements have been specified for public network exchanges and can be found in ITU-T specifications.

For the purpose of dimensioning signaling channels, it is useful to consider the sizes of different signaling flows (such as the setup flow or the full session flow) in bits and calculate, how long does it take to carry the flows over a given signaling channel of a given capacity. For example this analysis helps to validate how well does the design meet the technology requirements specified in the technology viewpoint. The analysis also helps to pass judgment on the applicability of the protocol to different network and service scenarios and to different environments.

Historically, networking people have paid a lot of attention to performance. It is a numeric requirement and therefore lends itself to mathematical analysis – a clear favorite for the engineering science. As Moore's law for computing power has been in

force since the 1960's and the networks have become faster, it has become easier and easier to meet given performance requirements. At the same time the expectations of the end users have risen. Poor performance compared to the competition may result in high cost for a given scenario.

We have said that strict performance requirements are typically in contradiction with all other nice requirements: the higher performance is required, the harder it is to achieve the other nice requirements (reliability, flexibility etc.). Therefore, over the years, the designers have gradually sacrificed performance requirements in order to make it easier to achieve the other nice requirements. For example the move from binary protocols to text-based protocols is an example of this trend.

Reliability requirements and design patterns

Reliability is another numeric requirement. Reliability is the probability that a system is operational at time t when it is known that it was operational at time $t = 0$. Methods of reliability engineering can be applied to signaling protocols' design. For example, one can break the system under design into components, find out or assign reliability values to the components, and assuming that the failures of the components are independent, calculate the total failure probability of the system. This kind of reasoning is called *reliability engineering*. It has been applied to signaling protocol design and implementations extensively over the past 40 years.

Given the VLSI circuits on a board, one can calculate the hardware failure probability of the board using information on the data sheets of the circuits. Given the boards that make-up a larger hardware block such as a control computer with a number of pre-processor cards, one can calculate the failure probability of the hardware block with a parallel/sequential component model of the block. A similar parallel/sequential arrangement of hardware blocks involved in an activity, such as a telephone call, can be used to calculate the failure probability of the activity. For the call, the probability of premature release (failure of an established call) and the probability of call setup failure are given in ITU-T specifications (Q.542 etc.).

If a given hardware design does not meet the requirement, one can add *hardware redundancy*. It makes sense to deal with the least reliable block first. Alternative forms of redundancy are *data redundancy* such a checksums and *redundancy in time*. Data redundancy sacrifices some transmission capacity, cpu-power and time of calculations for the sake of higher reliability. Redundancy-in-time uses several different protocol mechanisms for the sake of reaching the target reliability by sacrificing processing time, computing power and transmission capacity. Which type of redundancy to use, depends on the problem and the constraints like the required performance. In real-time functionality like signaling and call control some well-known data/time redundancy techniques that are applicable elsewhere, cannot be used. For example, in most cases it does not make sense to store call state onto disc storage for the purpose of recovery of the state in case of hardware failure. While the recovery would be taking place, the events in the environment would have overtaken the control system that would loose touch with the environment.

Reliability engineering in a network wide scale has not been paid extensive attention to. Most reliability models assume that the different types of failures are independent. Having made this strong assumption, the system level reliability is easy to calculate

once the component reliabilities are given. However, it is typical of *large networks* that the assumption of failure independence does not hold: often failures tend to escalate (a few years ago we witnessed the same kind of escalation in a Nuclear power plant in Japan).

Examples of how to formulate reliability requirements that are relevant for signaling protocols are:

- Probability of packet loss
- Probability of session failure (better break it to the next 2)
- Probability of premature release of a call
- Probability of call setup failure or misrouting

For example, given the probability of premature release of a call (a call being torn down without initiative from either the caller or the callee) of $2 * 10^{-5}$/min (Q.542) in a network node, one can take this as a system level requirement, break the system into constituent components (equipment blocks) and verify whether the requirement can be met. Redundant blocks appear in the reliability model in parallel (it is enough that one of them works for the system to be operational). All blocks that need to work for the system to be operational appear in a sequential reliability model.

The probability of failure of a duplicated block is $F_d = f_1 * f_2$, where f_1 and f_2 are the probabilities of failure of the duplicating halves of the system. Then the reliability of the duplicated block is $R = 1 - F_d$.

When we have a chain of blocks, i.e. when all the blocks in a sequence need to be operational for the whole system to work:

$R_c = R_1 * R_2 * \ldots * R_n$, where the left hand side is the system reliability and the elements of the product on the right hand side are the component reliabilities.

If the requirement is not met, one can identify the weakest link and replicate it or apply some replication scheme (such as N+1 of some sort). Verify again and carry on this process until the requirement is met with the help of sufficient level of redundancy and replication of operations.

Besides the failure probability one may be looking for high availability performance. This is different from reliability in the sense that availability allows repair while reliability does not allow it. Under the analysis of availability performance we talk about he *mean time to failure* (MTTF), *mean time to repair* (MTTR) and *mean time between failures* (MTBF). This kind of analysis is necessary for example for deciding whether hot-standby (MTTR $\rightarrow$ 0) or cold-standby (MTTR > 0) replication will be used.

Overall, the way to improve reliability is to introduce *redundancy*. Redundancy can be in terms of time, equipment or data. An example of redundancy in terms of time is to introduce acknowledgements into the protocol under design. Acknowledgements help to achieve a level of desired reliability of protocol operation at the cost of extra delay. Data redundancy comes for example in the form of checksums, parity bits or error correcting codes. Equipment redundancy may require certain kind of support for load sharing, load balancing or for recovery or restoration procedures. Restoration

procedures may introduce additional delay: once again we may trade delay for higher reliability.

Replication schemas

The choice of suitable replication schemas is guided by several considerations, like

- Avoiding single points of failure (it is desirable that the service of lots of users is not impacted by a single failure or that the whole system can not fail because of a single hardware failure).
- Reasonable hardware and software rearrangements, and upgrades and extensions of the hardware should be possible without breaking live service to customers in a public network. For example, it may be possible to use the extra hardware introduced mainly for high reliability/availability performance for the purpose of minimizing service disruptions while the software of a live node is being upgraded.
- It may follow from the first bullet that a hardware block must be duplicated. This can be a *hot-standby* arrangement where one block is active e.g. processing the live call traffic and the spare block mirrors the state of the active block in real time. In this case the target is that the hardware block *failover* does not disturb the live load in any way or only marginally (MTTR → 0).
  Alternatively, the spare block may be "cold" – i.e. it is not mirroring the live state of the active block but is ready to take over fresh load immediately. For example when there is only one spare block for N in-service blocks, it may be infeasible to mirror the call states of N to just one block.
- It may follow from the first bullet that it is sufficient to have some spare capacity to handle the load. E.g. when one block of N fails, it is taken out of service but the spare capacity of an extra block (or the rest of the blocks) is used to carry the whole fresh load. In this case, the load of the failed block may be lost.
- When all remaining blocks share the load, we talk about load sharing.

Considering reliability in node software

In real-time systems like telephony or more generally networking systems, the software designer must assume that anything in the environment of the module under design can fail at any time. This assumption costs a lot in terms of design effort but it is necessary in order to minimize the possibility of a deadlock situation. One very common method to follow this assumption is to set a timeout every time the program reserves a resource or starts expecting an answer from another software block. If the external events do not make use of the resource, update the timeout and release it in a controlled way, the timeout will guard the reservation and release the resource at expiration.

On the system level, we must have ways to collect the observations that something has failed, based on such indications, the system must locate the failures to blocks of hardware and software, automatically *isolate the failed part* and if an on-line spare is available *replace the failed block* with a redundant one (cold or hot standby). The system may also have *fault location functions* to locate the fault more carefully to a hardware unit that can be readily manually changed to a spare.

Reliability vs. correctness

In distributed system design the goal may be correctness (for example in banking, we must be ready to ensure at all costs (e.g. in time) that the account balance is always correct). Networking systems that must work in wide area in real-time are not optimized for correctness. Instead, the target is to create systems that work with high level of reliability where the maximum probability of failure is given but is more than zero. For example, it is probably impossible to create a distributed routing system for the global Internet that would be able to ensure that the routing tables of all nodes are always correct and no packet is ever lost because of the instant inconsistent state of routing tables. The difference between these two categories of systems: (a) correct and (b) highly reliable is huge in terms of cost and feasibility.

Reliability mechanisms in protocols

On this course we have learned that many reliability mechanism are used in signaling protocols on different layers.

The mechanisms include:

- Header checksums (e.g. IP)
- Frame or packet checksum (Ethernet, LAPB, UDP)
- Acknowledgements (many layer 2 protocols and (L3) signaling protocols)
- Selective acknowledgements (TCP)
- Message numbering (LAPD, LAPB, RTP)
- Message time-stamping (RTP)
- Buffering of sent messages (MTP in SS7)
- Resending (upon negative ack or gap in numbering)
- Windowing (for flow control to avoid buffer overflow at the receiver; LAPB)
- Forward error correction
- Redundant channels (MTP)
- Asynchronous feedback (RTCP)
- Etc.

These mechanisms may improve the reliability by several orders of magnitude and also have a significant impact on the perceived quality of service by the users.

Impact of failures

Failures can be classified based on their impact. For example, complete system failure, partial failure and reduced capacity. A nice feature of a system is *graceful degradation*, i.e. under failures the system maintain at least a part of its capability to serve the live load. A centralized system may be highly vulnerable to complete system failures while distributed systems have a natural capability of the complete failures being rare and graceful degradation under failures.

For an operator, high reliability with a reasonable cost is desirable because more failures mean more trouble shooting, more manual work and higher operational expenditure. How the amount of trouble shooting work grows as a function of the network size is an aspect of scalability. When the failures are visible to the customers, they may seriously undermine the operator's reputation on the market or leave the customers dissatisfied encouraging churn.

Historically, the move from analogue telephony to all digital exchanges and transmission led to a leap of improvement in reliability and consequently to reduced need of manual work and reduced operational expenditure.

Scalability

Scalability is the adaptability of the system to volume requirements. When service take-up is low, we need to be able to build small systems with low initial cost. Conversely, when service take-up grows, we must be able to meet the growth by adding capacity with a reasonable cost. As the business with a new system, service or technology picks up, the number of users, operators, other business actors, network nodes and the usage of the service/system/technology per user will grow. Scalability implies that the service/system or technology can easily adapt to these changes and as a result the investments already done do not need to be thrown away and replaced by new implementations of the same thing.

Scalability is a derived requirement in the sense that it combines performance, reliability and parallelization of execution. Often one also needs to look at the detailed parameters of the protocols. Depending on the system under design we are interested in different aspects of scalability. It is important to identify all the relevant scalability questions when you are designing a particular system. For example *scaling the system to*

- Any number of subscribers
- Any reasonable number of operators
- Suitability to different network topologies (large number of nodes, small number of nodes, high or low degree of nodes).
- Low level of traffic – high level of traffic per user; adaptability to different patterns of arrival of traffic
- Short flows
- Long flows
- Any number of applications
- Payload size/total service flow (how much signaling and header overhead)
- Suitability for mobile use (how often and how far the users move).
- Suitability for battery powered devices.
- Power consumption as a function of traffic, number of nodes or users
- Etc.

Scalability is a strong invariant in network system design. Over the past 30 to 40 years it has always been important. It strongly correlates with system cost and with what the system can be used for, i.e. to the earning potential of the technology.

Flexibility requirement and design patterns

Flexibility is the adaptability to *new requirements* and *new use scenarios*. This requirement relates to the *time-to-market* aspect of new technology. Over the past 30…40 years, flexibility has become more and more important. Networking system engineers have been willing to sacrifice all other requirements (in particular performance and even reliability) for the sake of higher perceived flexibility. The reason is that in ICT business the experience is that “winner takes all”. I.e. who is first

to the market with a new technology, product or feature will earn the most while the latecomers to the market have to be happy with what they can get.

For the operators, the flexibility of the technology they buy and use relates to their need for *differentiation*. In the business strategy classification (M. Porter), differentiation is the most popular strategy because it allows for many winners (the other alternatives are *niche strategies* and *cost leadership*).

Flexibility is a non-numeric requirement and thus difficult to formalize in any way. Also, let us be realistic: since we do not know what the future requirements might be or what possible future scenarios are relevant, how can we make the technology ready for adaption to these new requirements or scenarios? Let's admit, we are talking about guessing and at best about the *art of design* rather than about any kind of exact science.

However difficult it is to formalize flexibility, systems that cannot be adapted to new requirements or are hard to adapt to new scenarios will fail on the market. Also, it is typical of ICT systems that the vendors and users will keep modifying them so long as it is possible to do so to meet new needs. This kind of adaption to new uses and scenarios allows leveraging the already gained positions on the market etc.

Design patterns

How does one then improve the adaptability to new requirements? The classical approach is to go for a *modular design*. I.e. break the functionality into compact functional entities minimizing the interactions between the entities and defining clear interfaces between the entities. Moreover, if the classical OSI-model leads to a stack of protocols one above the other, i.e. a fixed overall structure, some modern designs are *frameworks* that try to make it possible to combine the components of the design in different ways for different use cases.

Over the past 10 to 20 years the idea of modularity has found a fresh implementation in protocols. Many protocols (e.g. Diameter and SIP) have been designed with the structure: *base protocol plus extensions* where the base protocol is supported by all protocol-speakers but the extensions are optional. This is a nice way of allowing for tailoring to particular use cases and designing for differentiation.

In signaling protocols the classical dominant design model is that of *communicating state machines*: each protocol is designed as a set of (extended) finite state machines. In such a design, it is important to take care that all resource reservations are guarded by timeouts and that the state tends to IDLE with all reservations released unless there is a clear reason not to move to IDLE. In this model the way to react to an incoming message depends on the *history of the service flow* and the content of the message.

An alternative design pattern for protocols is the *client-server model*. In this model the reaction to an incoming message is not supposed to depend on the history of the service flow but only on the content of the message, some prior configuration and possibly some prior events that are seen as independent of the current service flow (think about your bank account that is maintained in an SQL DB).

This model is easier to make to work if it is suitable for the problem at hand. User to user communication is a peer-to-peer phenomenon, so one can argue that it fundamentally points to the direction of communicating finite state machines. SIP is an attempt to use the client-server model in signaling. SIP-kind client-server does not really follow the restrictions that are usually seen inherent to client-server. Rather we can see it as an example of the engineer's desire to stretch existing solutions to new uses in ICT. In this case the result is less than minimal signaling flows with the gain of possibly clearer model of processing transactions.

Extensibility

The requirement of extensibility applies to most protocols, in particular signaling protocols. It is a consequence of the need to adapt to new scenarios. The protocol standard may set rules to the future designers on how in particular the future modifications can change or extend the protocol and what things can not be changed (for example ISUP since 1992). Protocols may carry a *version* number (e.g. SIP, IP) or versioning may take place on the level of feature packages (MAP), they may have *capability negotiation* (e.g. Diameter) that allows agreeing by communicating nodes on the procedures that are supported. Protocols may have mandatory and optional messages and information elements. Protocols may have *fallback routines*, the idea being that if the neighbor does not support a new version, the nodes can fall back to the behavior specified for the previous version. Protocols may have to carry *compatibility bits* that define what the receiver should do if it does not understand some piece of new information in a new version.

These kinds of extension rules are necessary because it must be possible to update the protocol version in a live network that is carrying live traffic without disturbing the users. The updating must take place one network node at a time. Updating a large network node may be a rather heavy operation that takes many hours. It may be necessary to carry it out during the night hours because testing of the changes may require a restart or some other disturbance for a short period of time. In any case upgrading nodes in a public services network requires more effort, planning and care than similar updates in corporate networks. If updating one node takes a working day, updating all nodes in 500-node network takes about one year for two teams working in parallel. This kind of work is a part of operational expenditure of the operator.

Because of typical extension rules it may be not allowed to change

- Identification data
- Mandatory information elements
- Encoding rules
- Semantics of any information elements
- Order of information elements (depends on coding style)
- Etc.

It may be allowed to

- Extend the protocol with new optional information elements in the end of the existing messages
- Extend existing optional information elements at the end
- Add completely new message types.

The classical *extension rule on a network level* is that new technology must be adapted to the existing network with its legacy technologies. For example, it was

necessary to make digital mappings for analogue signals of analogue signaling systems in PSTN and new types of trunk signaling (TUP, ISUP SIP) have had to be adapted to support some features what were originally invented for analogue PSTN. This rule is sometimes difficult to adhere to. If a new technology, communication method etc. is not compatible with the existing network, this may lead to slow adoption of the new technology: the Metcalfe's law of network value (value is proportional to the square of the number of users) needs to ramp up from zero rather than from the existing subscriber base of some Billions.

Changing importance and interactions of requirements

Over the past 30…40 years the importance of flexibility has been growing at the expense of performance in particular. The importance of scalability has been stable except that some aspects of scalability have been sacrificed for higher flexibility. With the move to Voice over IP, protocol designers have sometimes moved from binary protocols to text based protocols (e.g. SIP and SDP). We have seen that for call signaling protocols this sacrifices scalability for short flows for flexibility while the conversion of text to binary and binary to text takes a lot of memory and CPU-cycles, i.e. sacrifices node performance for the sake of flexibility.

There are two camps in terms of reliability: (a) the traditional ITU-T and telecoms approach: go for high reliability from the start and (b) the IETF/Internet approach: create some working code and improve gradually plus rely on Moore's law to help with performance bottlenecks.

One should also note some rarely discussed interaction of the different requirements:

Operator's business needs vs. flexibility:

The operators have been calling for technologies that allow *fast service implementation* since 1980's. This is motivated by the need to differentiate and meet the user needs as quickly as possible. One should however notice that business differentiation takes place only if the other operators cannot copy the new thing quickly. Also, it is important that there are no zero-cost alternatives. Usually the vendors cannot afford to design new features for exclusive use by a single customer. So, once the new software needed for the new feature is designed, it quickly becomes available for all operators. There is a strong tendency that one can differentiate with the newest features but the old ones quickly become part of the standard package. Under this kind of race, it is however clear that even if the end-result is no differentiation, a player in the market cannot choose not to compete. That is a sure loosing strategy.

When the technology development focus has moved to Voice (Multimedia) over IP, the operators have insisted on a complicated design that ensures *operators' control over what the subscribers can do*. So, retaining control is even more important than flexibility. This is what one should expect because if there is no control, it may be difficult or impossible to earn money. Alternatively, one can argue that the operators should think about new earning models.

Flexibility vs. reliability

Extreme flexibility of a protocol raises the objection that protocols are supposed to be designs that meet the requirements of well understood problems. If a problem is well

understood, why should it be extremely easy to modify the design? Also, if it is very easy to do the modifications, probably many vendors will try to make use of this option (if not, why the flexible design?). The result is that there will be numerous modifications of the same thing on the market. What guarantees are there that all those modifications are interoperable? If they are not interoperable, flexibility in terms of building multi-vendor networks is lost. Or alternatively, we can see that the operation of multi-vendor networks is unreliable – most of the time they may interwork but sometimes non-compatible interactions take place.

Such dilemmas set the limit on how far is it possible to take the requirement of flexibility.