Quality of Service considerations for Network Based Defence

Frank Trethan Johnsen, Trude Hafsøe and Ketil Lund

Forsvarets forskningsinstitutt/Norwegian Defence Research Establishment (FFI)

26.02.2007

FFI-rapport 2006/03859

898

ISBN 978-82-464-1141-5

Keywords

Tjenestekvalitet / Quality of Service

Mellomvare / Middleware

Nettverksbasert forsvar / Network Based Defence

Abonnementstjenester / Publish/subscribe services

Nettverksmellomledd / Proxy servers

Tjenesteorientert arkitektur / Service oriented architecture

Approved by

Ole-Erik Hedenstad	Prosjektleder/Project manager
Vidar S. Andersen	Avdelingssjef/Director

Sammendrag

Denne rapporten diskuterer konsepter, og fokuserer på applikasjonsnivåløsninger som vil være nødvendige bestanddeler av Nettverksbasert Forsvar (NbF) i framtiden. Vi fokuserer på tjenestekvalitet (Quality of Service), og hvordan man kan oppnå dette ved bruk av mellomvare. Videre presenterer vi de innledende eksperimentene vi har gjort med mellomvare over trange kanaler, og diskuterer viktigheten av kommunikasjonsoptimalisering – bruken av abonnementstjenester (publish/subscribe) og hvordan mellomledd i nettverket (proxies) kan videre effektivisere kommunikasjonen.

English summary

This report discusses concepts, and focuses on application level solutions that will be important building blocks in Network Based Defence (NBD) in the future. We focus on Quality of Service (QoS), and how to achieve it with the help of middleware. The preliminary experiments we have performed with middleware over disadvantaged grids are discussed, along with the importance of optimizing the mode of communications itself. In this context we present relevant standards for publish/subscribe services, and discuss how intermediaries in the network, so-called proxies, can make communications more effective.

Contents

1 Introduction

Quality of Service (QoS) plays a very important part in fully realizing Network Based Defence (NBD), the Norwegian equivalent of Network Enabled Capabilities (NEC). The NBD vision implies an information infrastructure that supports prioritized access to information, services and communications resources from the strategic level, down to the tactical level where communication resources are usually scarce. Thus, without effective QoS management, the NBD vision will be very hard to accomplish.

The goal of this report is to provide an overview of what QoS is, together with a comprehensive presentation of the most important mechanism for QoS management in distributed systems, namely middleware.

The NATO NEC Feasibility Study (NNEC FS) presents a discussion of technology and NEC, focusing on the needs of future interoperable military communications. An information infrastructure supporting NEC will have to allow for communication across system and national boundaries while at the same time taking legacy systems into account. This leads to a requirement for a flexible, adaptable and agile information infrastructure which can support all the communication needs of national forces, and at the same time support interoperability. Due to the limited resources available in military communication networks, this can only be achieved by having information systems that support end-to-end QoS.

The NEC vision of always making sure that information reaches whoever needs it means an increase in the number of messages transmitted between users and across system boundaries. This in turn means an increased need for QoS: the limited resources available need to be utilized so that they best serve the needs of the system users. NNEC FS lists a number of aspects which are essential to the process of creating a common communication infrastructure, and *ranks the implementation of a QoS framework and policy as the second most important aspect of this process.* In addition, the NNEC information infrastructure will need to support both traditional communication services, but also need to anticipate new services, like interactive multimedia and voice over IP. These data types have different transmission requirements compared to textual messaging, and must be handled accordingly.

The Network Centric Operations Industry Consortium (NCOIC) has, in addition to making its own technology recommendations, performed a review of the NNEC FS. One important comment from this review is that even if the NNEC FS identifies the need for end-to-end QoS, there is little discussion regarding how end-to-end QoS should be achieved. Signalling QoS information between systems and across national borders is needed, and the use of a standard intra-domain QoS protocol is recommended.

A vision of the future is presented in [21]: *"How are ad hoc networks likely to evolve? ... Military ad hoc networks will have higher capacities and support multimedia applications, be more adaptive, stealthy, and evolve toward a system where all battlefield elements, mobile or stationary are multimedia-networked*". Achieving this will be NBD in its final stage (there are three stages of NBD, see [23]). Getting there will not be easy, however. The first step towards NEC/NBD is to integrate legacy systems from the strategic to tactical level into a common network. For such integration the modular concept from Service Oriented Architectures (SOA) is essential. Each legacy system can be viewed as a separate module that needs to be interconnected with others. In order to get the different modules to cooperate one needs a common standardized means of communications between them. This infrastructure should preferably be built using standards and commercial off-the-shelf (COTS) products. Middleware in its simplest form would be software used to "glue" various COTS together to form an interoperable system built on SOA principles. In addition to connecting various COTS, the middleware must also provide for QoS. It could also have other tasks, such as implementing single sign-on and security measures, but that discussion is beyond the scope of this report.

The report is organized as follows: In chapter 2 we discuss various aspects of QoS in a military context. In chapter 3 we present a simple communications scenario that we refer to in examples throughout the report. Chapter 4 introduces the concept of middleware, and provides a discussion of challenges when employing such technology for military use. The chapter concludes with a discussion of our preliminary experiments. In chapters 5 and 6 we discuss Web Service standards relevant to Publish/Subscribe services, and present how we foresee the use of such technology in tactical systems. The latter chapter also discusses video caching and adaptation techniques, which will be relevant for the future of multimedia-enabled military networks. Chapter 7 summarizes the report.

2 Quality of Service

In disadvantaged grids it is important to utilize the scarce resources as efficiently as possible. Different types of data will traverse the same network, and compete for these resources. Not all data is equally important, and since it may not be possible to satisfy the needs of all users simultaneously, one should categorize the information according to priority. For example, important orders should be relayed quickly through the system, whereas routine information can be delayed (or in some cases even discarded). End-to-end QoS is necessary to achieve such prioritization of information. However, this is only one part of the QoS puzzle, as there are also many other aspects that play a part. Our previous research regarding NBD and Web Services [4] has indicated the lack of current suitable QoS solutions, and the need for QoS enabled middleware. In this chapter we will discuss various aspects relating to QoS, framing them in a military context.

2.1 Background

QoS is a central issue when discussing all types of systems design, ranging from multimedia entertainment systems built on Internet technology to tactical military communications systems.

The concept of QoS was first used within the area of data communication, in order to describe technical characteristics, such as delay and error rate. With the advent of multimedia applications, the QoS concept has been extended to cover end-systems as well, to enable an end-to-end control of the quality level. Consequently, QoS concerns all parts of a distributed system: client system, network, and server. According to [28], there are five categories of QoS-parameters, and these are shown in Table 2.1.

Category	Example parameters
Performance-oriented	End-to-end delay and bit-rate (bandwidth)
Format-oriented	Video resolution, frame rate, storage format, and compression scheme
Synchronization-oriented	Skew between the beginning of audio and video sequences
Cost-oriented	Connection and data transmission charges, and copyright fees
User-oriented	Subjective video and sound quality

Table 2.1: The five categories of QoS-parameters

QoS-parameters from all categories except the cost-oriented one are relevant for a military NBD system. For the user, the user oriented parameters are the ones that matter, but the user oriented parameters are in turn dependent on the performance-, format-, and synchronization-oriented parameters. For instance, to perceive the subjective image quality of a video as being satisfactory,

the video must be coded in a format and resolution that allows a sufficiently high image quality. Next, the system must be able to provide a sufficiently high bandwidth to support the bandwidth requirement of the video.

The performance-oriented QoS-parameters are typically associated with the lowest layers (i.e., closest to the physical resources) of a system, and therefore impact parameters in most other categories. In other words, mechanisms employed at lower layers affect the services offered to higher layers.

The QoS semantics is an important part of QoS management. In [29], five types of QoS semantics are defined:

- *Best-effort QoS*: The weakest type of obligation. All components in the system do their best to meet the requested QoS level, but no guarantees are given.
- *Guaranteed QoS*: The required QoS level is guaranteed by the service provider, and is supported through allocation of the necessary resources.
- *Compulsory QoS*: Like guaranteed QoS, the QoS level is guaranteed by the provider, through resource allocation. In addition, the QoS parameters are monitored, and if the requested level can no longer be sustained, the connection is terminated.
- *Threshold QoS*: This is similar to compulsory QoS, but if the level can no longer be sustained, the user is notified, instead of terminating the connection.
- *Maximal QoS*: An upper limit to the QoS is provided, and the QoS level should not exceed this. This semantics is typically used if there is a cost attached to the QoS level, and the user does not want to pay more than a given price.

QoS semantics must be selected based on the user's personal preferences and perhaps earlier experiences with the system in question. Also, QoS can be chosen administratively, for example by assigning QoS to different user roles, so-called role based QoS. For the network subsystem, an important consequence of these semantics is the need for admission control and resource allocation. That is, for guaranteed, compulsory, threshold, and possibly maximum QoS, a certain share of the bandwidth and buffer space is reserved in order to fulfil the QoS level agreed upon. The result is either admittance of the query and subsequent reservation of resources; or the query is rejected, meaning either that the QoS level must be renegotiated, or that the client must be rejected due to lack of system resources. In addition, except for best-effort QoS and guaranteed QoS, all semantics require that the actual QoS level delivered by the storage subsystem is monitored. Finally, we make a distinction between statistical and deterministic guarantees. For real-time requirements, a deterministic guarantee implies that the requested data always will be delivered on time. However, this level of guarantee is expensive, since we must reserve resources according to worst case requirements. For example, when variable bit-rate multimedia data is present, this means poor utilization of bandwidth. A statistical guarantee means that a certain percentage of all requested data will be on time. For instance, for real-time requests, a statistical guarantee could state that 99.5% of all information will be delivered on time. The difference between the two guarantee levels lies primarily in the way admission control is performed: while a deterministic guarantee requires admission control and resource reservation according to worst

case requirements, a statistical guarantee is generally based on using some sort of average resource requirements. Thus, more clients can be admitted, and the resource is utilized better. The price to pay for this increased utilization is occasional resource shortage, since there may be times with peak load, where the resource requirement exceeds the amount reserved. However, the users of a statistical guarantee service are aware of this fact, and must be able to handle such violations of the service level agreement. For instance, deadline violations can be handled through increased buffering.

In [30], three types of deadlines are identified, namely hard, firm, and soft deadlines. Violating a hard deadline is catastrophic, and must never occur. With a firm deadline, a violation is not disastrous, but the data has no value once the deadline is passed. For soft deadlines, the requested data may still be of some value after the deadline, but this value is monotonically decreasing, and at some point it reaches zero. Depending on how delayed data is handled, the deadlines in multimedia playback are either firm or soft: if delayed frames in a video are dropped, the deadlines are firm, but if delayed frames mean that the entire playback is delayed, then the deadlines are soft.

2.2 QoS and NNEC FS

QoS is a term that encompasses many different aspects, *Figure 2.1* illustrates this. Traditionally, QoS has been used to describe quantifiable network parameters such as bandwidth, latency, jitter, and packet loss. In an NBD setting we need priorities not only on the network level, but also from end-to-end in the system and on the application level. We use the term QoS in a much wider sense, and refer to network related parameters as *network level quality of service*. The network level QoS only covers the performance-oriented QoS parameters.

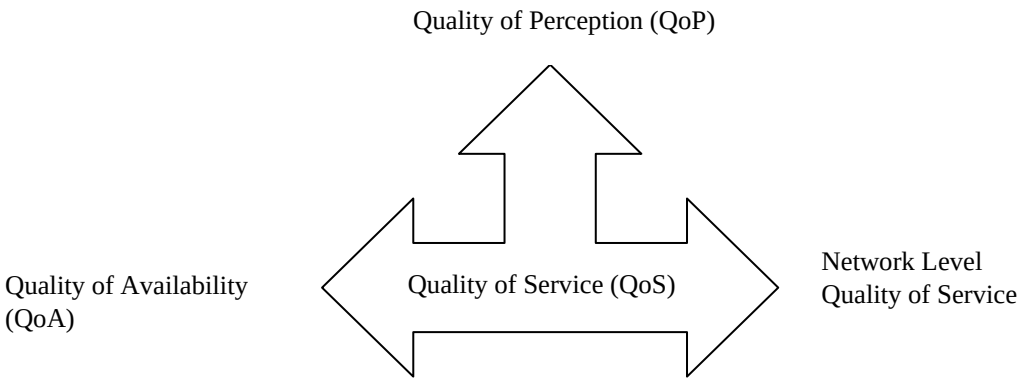

Figure 2.1: QoS in a military setting focuses on QoP, which in turn must be quantified, expressed in quantitative terms and used with network level QoS and QoA to achieve the overall quality goal.

The NNEC FS has suggested that NEC should perform all communications with “Everything over IP” (EoIP). The Differentiated Services (DiffServ) [7] framework has been identified as a good solution for network level QoS by several NEC related research projects, two being TACOMS [5] and INSC [6]. Based on this we anticipate that IP with DiffServ will be considered

for use in the NBD solution.

In addition to the network aspects, we also have to consider the needs of users and the fact that some system components may fail and become unavailable. These issues refer to user level quality of service, so called Quality of Perception (QoP) [2], and Quality of Availability (QoA) [3]. QoA is a measure for availability; replicating services and data in order to maximize availability and minimize system downtime. QoP, on the other hand, is a qualitative measure, and not quantitative like network level QoS and QoA. QoP is not fulfilled if the user is not satisfied with the system's performance. Thus, the users' needs must be analyzed and quantified, so that the QoP requirements can be mapped to physical parameters [1]. NNEC FS defines quality of service as *the user's satisfaction with the service*, but the study also mentions the importance of quantifiable parameters. As a consequence, several aspects together form the overall QoS.

An established and more descriptive definition of QoS than the one given in NNEC FS is from [28], and appropriately summarizes this discussion:

"Quality-of-Service represents the set of those quantitative and qualitative characteristics of a distributed multimedia system necessary to achieve the required functionality of an application."

2.3 QoS: End-to-end and Cross-layer

Essential to NBD is the concept of end-to-end QoS, which in turn requires employing cross-layer signalling. This means that QoS must be considered at all layers of the OSI model, and that QoS information must traverse these layers. To achieve the end-to-end property needed in NEC, QoS information must also be allowed to cross both network and national boundaries. There already exist QoS mechanisms that can be used on the transport layer and below, and thus we focus our research efforts on the application layer and issues regarding cross-layer QoS signaling. Having IP as a common protocol and assuming DiffServ as the network level QoS framework, we focus on two application level solutions in this report; middleware and proxy servers. The task of these solutions will be to provide for QoS at the application level, and map the demands to and from the Type-of-Service (TOS) field in the IP header, enabling cross layer QoS signalling.

However, it is important to remember that to achieve true NBD with support for interoperability with other NATO nations, one needs to have a standardized means to signal QoS demands between systems belonging to different nations. This is a requirement to ensure that the information flow is given the proper QoS resources all the way from end-to-end in the systems. In this report, however, we merely present techniques for achieving various aspects of QoS, and do not attempt to address standardization issues.

3 Scenario

This report discusses QoS and its application to military communication systems, illustrated by the simple scenario[1] in *Figure 3.1*. The goal of this chapter is to give an overview of which forms of interactions we can anticipate, and also which kind of communications technology that is available on various levels in the military organization. We focus mainly on tactical communications in this report. Communication on the tactical level is characterized by wireless networks with low bandwidth, and possibly high delay and high error rates and frequent disconnections. As a consequence, these networks are often called *disadvantaged grids*. Because of the severe limitations of disadvantaged grids, a solution that works over disadvantaged grids and supports NBD deployment can be used for wired networks, where resources are more plentiful (although it is likely that one will provide more functionality in these networks due to the increase in available resources).

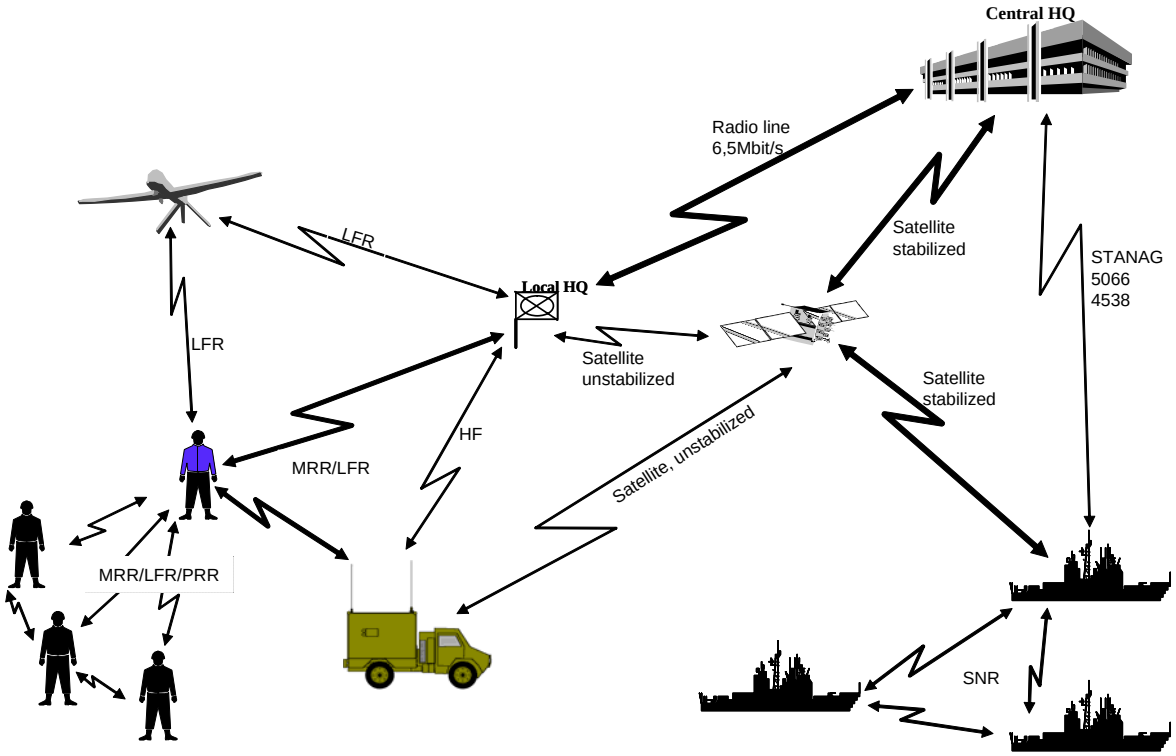

Figure 3.1: A simple communications scenario

Typically, the communications between the central headquarters (HQ) and a local HQ will use a radio line capable of transmitting 6.5 Mbit/s. One can also communicate over satellite links with stabilized or unstabilized antennae. Stabilized antennae require infrastructure, so the unstabilized ones are usually used in the tactical communications since they are more portable. Unstabilized

[1] This simplified military communications scenario focuses on technology aspects. It is based solely on information obtained from project 869 [24], with the exception of part of the information about the UAV which is from [25].

antennae yield a much lower bandwidth than the stabilized ones do, typically 2.4 Kbit/s.

The central HQ can communicate with vessels over HF radio, typical systems being STANAG 5066 and STANAG 4538. The radio has a number of channels to choose from, and will have to find one that works by trial and error. Which channels are available and usable will vary with the weather and the time of day. Because the radio needs to probe for channels, the link establishment phase can be very long – up to 30 seconds in the worst case. When the link is established communications can commence; typically with a bandwidth between 150 bit/s and 9.6 Kbit/s. STANAG 4538 is a newer standard, but it has the same bandwidth and intended use as STANAG 5066. The difference is in the way the sender and receiver side synchronize with each other to establish the link: They use GPS time for this, which enables them to find a suitable channel much quicker.

Communications between vessels can be performed with Sub Network Relay (SNR), which has a bandwidth of approximately 64 Kbit/s. A vessel with a stabilized satellite antennae will function as a gateway between the rest of the vessels and the HQ. This gateway can also use STANAG 5066/4538 for communications, in which case it should also function as a proxy for the vessels in the SNR network – this is important to adapt the data flow to the low bandwidth available on the HF link.

The HQ may communicate with land-based forces in a similar, hierarchical manner using radio or low bandwidth satellite communications with local HQs. Vehicles can communicate directly with the HQ via satellite and with a local HQ over High Band UHF (HB-UHF).

Inter-vehicle communication can be done using the multi role radio (MRR), which offers a maximum bandwidth of 64 kbit/s. It is usually configured to use a bandwidth of 2.4 kbits/s in order to increase the communication range. HQs and vehicles can communicate with squad leaders via light field radio (LFR), which is the small, portable version of MRR. In contrast to the low bandwidth available when using MRR or LFR, HB-UHF has a bandwidth of around 300 kbit/s, but a much shorter range. This means that if one can dynamically choose between the two, one can utilize the bandwidth of HB-UHF as long as it is available, and then fall back to MRR/LFR when the range is too long for HB-UHF. This would require the middleware to support multiple networks and a dynamic choice thereof.

Squad members communicate with each other using a personal role radio (PRR), which offers a bandwidth of 38.4 kbit/s. Only the squad leader will communicate directly with a vehicle and the local HQ. Thus, the squad leader acts as an intermediary for the other squad members. Intermediary nodes, so-called proxy servers, are important for performance issues, and should preferably be placed in the transition point between networks. See chapter 6 for further details about proxies.

The UAV needs a bi-directional communications capacity for receiving control orders, and for reporting its position back to the controller. Being equipped with one or more cameras, the UAV

can send a video stream back to the controller from the camera of choice. Today, this stream is sent using another transmitter, and is sent as an analogue signal. The video is captured and stored on the computer used by the controller, and can later be played back again and again as needed. There are two major problems with this:

1. An enemy can intercept the analogue unencrypted video feed and see what the UAV is interested in.
2. HQ can not see the video feed until the UAV operator returns with the computer which captures the feed, possibly losing valuable time that could be spent making tactical and strategic decisions.

To overcome the first problem, one needs to use a solution which digitally codes and encrypts the transmitted video. Security is beyond the scope of this report, so we will not dwell on that. The second issue should be addressed in NBD by allowing the computer controlling the UAV to be a part of the information infrastructure. Admittedly, the video signal could be relayed from the controller to the HQ, but this would still leave the first problem unsolved.

Furthermore, by allowing the video to be coded in a manner that allows it (or possibly a lower quality version of it) to be transmitted back to the HQ immediately, one can maintain the necessary information superiority the situation requires. The full quality video can still be stored on the computer, and can be played back at a later time. Video quality issues are discussed in section 6.2.

Table 3.1 gives a summary of the bandwidth offered by the various communications solutions.

Link Type	**Max bandwidth (Kbit/s)**	**Expected bandwidth (Kbit/s)**
High Band UHF (HB-UHF)	300.0	300.0
Sub Network relay (SNR)	64	64
Personal role radio (PRR)	38.4	38.4
STANAG 5066 / STANAG 4538	9.6	0.15 – 9.6
Satellite (unstabilized antennae)	2.4	2.4
Multi role radio (MRR) / Light field radio (LFR)	2.4	1.0

Table 3.1: Maximum and expected bandwidth for communications on the tactical level.

4 Middleware

According to [26], middleware is defined as a software layer between application and operating system. This is illustrated in Figure 4.1.

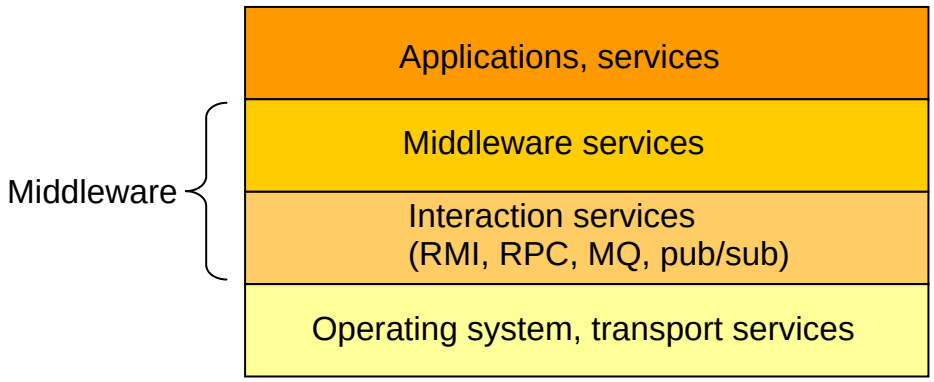

Figure 4.1: Middleware as a layer between applications and operating system

Middleware is usually viewed as a class of software technologies designed to help managing the complexity and heterogeneity found in distributed systems. However, there is no general consensus on the precise separation between these areas, neither on the required functions and services. In general, the separation will vary with time, as common application functionality is better understood, such that it can be standardized and moved into the middleware as a service.

The goal of middleware is to offer distribution transparency, i.e., to hide the consequences of distribution, with respect to things like location, access, concurrency, replication, errors, mobility. A common programming abstraction is offered, spanning a distributed system. Consequently, the middleware offers building blocks on a higher level than the APIs and sockets offered by the operating system. In other words, the middleware encapsulates general solutions on tasks that appear repeatedly in distributed systems.

4.1 Existing middleware

The notion of middleware was first used in connection with the introduction of Remote Procedure Call (RPC) and message queuing (MQ). These were technologies aimed at hiding the lower levels of communication, in order to simplify the work of the programmers, and they represent the first generation of middleware.

The second generation of middleware was based on distributed object technology. Instead of a client/server-model, a client/object-model is used, and objects are accessed through interfaces. The abstraction that is offered is that a remote object can have its methods invoked as if the object was in the same address space as the caller (this is called location- and access transparency). Examples of second generation middleware are Java Remote Method Invocation (RMI), Microsoft (Distributed) Component Object Model (COM/DCOM), and OMG CORBA.

There is, however, an important problem of this second-generation middleware, namely the presence of implicit dependencies. In other words, it is necessary to have a lot of information about the environment in which the application will run: How is the application deployed, which services will be available on a given server, who will activate my objects, who will manage the life-cycle of my objects, and so on. In the third generation of middleware, we see the transition to component-based technology, in order to alleviate this problem. Ideally, the components in a component-based system should only have explicit dependencies, which in turn guarantee *safe deployment*. This is a guarantee that if the dependencies are satisfied, the component will behave as specified. Consequently, the component can be reused by any application needing its functionality.

Middleware platform	Component model
OMG CORBA v3	CORBA Component Model (CCM)
Java 2 Enterprise Edition (J2EE)	Enterprise Java Beans (EJB)
Microsoft .NET	COM+

Table 4.1: Commercial third-generation middleware

Three component technologies are widely known, both in industry and the research community: CORBA component model (CCM), Enterprise Java Beans (EJB), and Common Object Model Plus (COM+). For each component technology there are corresponding groups of middleware platforms; Common Object Request Broker Architecture (CORBA), Java 2 Enterprise Edition (J2EE) [27] and .NET Framework.

The CCM is a language independent component model for the server side, supporting implementation, management, configuring, and deployment of CORBA applications. Basically, it is a container framework offering implicit middleware for security, transactions, persistence and event-based communication.
As for the J2EE, EJB is strictly speaking only one of several component models in Java:

- Java applets: downloadable light-weight components
- JavaBeans: client-side components
- EJB: component architecture for server-side components (integrated in containers)
- Servlets: server-side applets

COM+ offers a component object model based on the principles of binary encapsulation and binary compatibility. The former means that it is not necessary to recompile the client even if the server object is modified, while the latter means that client- and server objects can be developed in different environments and using different languages. The fundamental mechanism used for achieving binary encapsulation and compatibility is to separate between interface and implementation.

4.2 The problems of existing middleware

There are two main problems connected to existing component-based middleware, namely QoS management and tactical networks. In this section, we look closer at these two problems.

4.2.1 QoS

The main problem with respect to QoS is that today's middleware offers only *functional* safe deployment. In other words, although a given component is guaranteed to function as advertised, there are no guarantees as for how well it will perform its functions. For simple functions such as a file transfer, this may not be that important, but for an application that is sensitive to QoS; this may have a considerable impact on the usability of the application.

When a programmer designs a component, she will make assumptions about the environment in which the component will run, including assumptions about available resources. However, at design time, it can be hard to make any such assumptions, as the actual resource availability is usually not known until run-time. Consequently, the behaviour of the application becomes hard to predict.

Furthermore, to be able to maintain a given QoS, we need QoS management. Today, this management is usually hard-coded in the application (i.e., the components). This means once again that one has to make assumptions about the environment, which in turn limits the possibility of reuse. In addition, even if the component designer is able to predict the QoS, how can this be communicated to the user of the component? Finally, how do you predict the end-to-end QoS of an application, based on the QoS characteristics of the individual components?

A possible solution to these QoS-related problems is to introduce *separation of concerns*. This means that the application logic and the QoS management are handled separately, which in turn means that one has a much better opportunity of postponing decisions until one has sufficient information available. Admittedly, this is not possible with today's commercial middleware.

4.2.2 Tactical networks

A tactical environment is characterized by small and diverse platforms, and a highly dynamic environment, where both resources and services come and go continuously. This means that both the middleware and the application must be adapted to the given environment.

In such an environment, it becomes particularly difficult to predict the correct configuration of application and middleware during design time; information like what platform is used, and which resources are available and in what quantity, is only available at run-time. Thus, a number of configuration decisions can only be made at run-time. Today's middleware, on the other hand, only supports design-time configuration.

The dynamicity of tactical environments also necessitates the ability of both application and middleware to be reconfigured as resource availability fluctuates. In general, there are four types

of reconfiguration that can be performed, both on an (component-based) application and on the middleware itself:

- *Component-internal*: The application (or component within the application) handles the adaptation or reconfiguration internally. This is the traditional solution, which can be seen, for example, in Windows Media player.
- *"Turning knobs"*: Parameters within the application or component are adjusted externally. One example is the middleware adjusting the buffer size of a streaming application, in order to compensate for increased jitter in the network.
- *Replace components*: In this type of reconfiguration the implementation of a component is replaced. For instance, a compression component could be replaced with one that is less resource demanding (but slower).
- *Change composition*: The last, and most comprehensive, type of reconfiguration is to change the actual composition of an application. For instance, if the bit error rate of the network increases, a Forward Error Correction (FEC) component could be inserted into the application (on both client and server side), in order to compensate for the increased number of packet errors.

For all four types of reconfiguration, it is a requirement that safe reconfiguration is ensured. In other words, the application must be transferred from one stable configuration to another.
To summarize, in a context of QoS-sensitive applications, the most important requirements to a middleware are

- Safe deployment, also with respect to QoS.
- Run-time configuration, since much information with respect to environment and resources is not known until the application is started.
- Dynamic reconfiguration, since a tactical environment implies constantly changing resource and service availability.

4.2.3 Middleware and Web Services

Although Web Services can be seen as yet another middleware, there are differences. One difference is that while other types of middleware are typically used within a local domain, Web Services are used as middleware, to connect such local domains over the internet. In other words, they function as entry points into local information systems. Furthermore, Web Services can be viewed as an attempt to standardize middleware platforms with respect to language (XML), interfaces (WSDL), business protocols, and properties and semantics [14].

Traditional middleware face several problems that Web Services have the potential to solve. For instance, for cross-organizational interactions, the problem is where to place the middleware and who should control it [14]. There are also problems connected to long-lasting transactions and crossing of trust domains.

However, there are also a number of challenges associated with Web Services, indicating that it is not necessarily given that they will become *the* standard for all purposes when realizing SOA. In particular the lack of standards that everyone agree upon is a problem, and we currently have

several competing standardization efforts (OASIS, W3C, WS-I). Furthermore, the use of XML and the SOAP protocol means that the communication overhead is relatively extensive, a problem we have already identified in [19]. In disadvantaged grids it is important to reduce the amount of data traversing the network links. The available resources must be used optimally to ensure timely delivery of relevant information. Using Web Services over disadvantaged grids requires some adaptation to work. In our previous research, we have experimented with various compression algorithms to reduce the inherent overhead in XML message exchange, and we concluded that data compression is one of several means necessary to make Web Services work over disadvantaged grids. In disadvantaged grids the network is the limiting factor and not the processing capacities of the nodes, so compression is beneficial and should definitely be used. A key point for NBD is interoperability with other nations. The choice of compression algorithm should be standards based, so it is important to keep an eye on the developments in industry. Standardization efforts regarding binary XML are being undertaken by W3C at present [20]. Web Services are further described in chapter 5.

4.2.4 Preliminary experiments, static middleware

In a military context the middleware will have to support multiple carriers. The middleware should run on all systems, something which will require different options for different networks. Figure 4.2 illustrates this concept, where a common middleware can choose between MMHS (STANAG 4406) for communications over disadvantaged grids, regular HTTP/TCP for communications in strategic (Internet technology based) systems, and others.

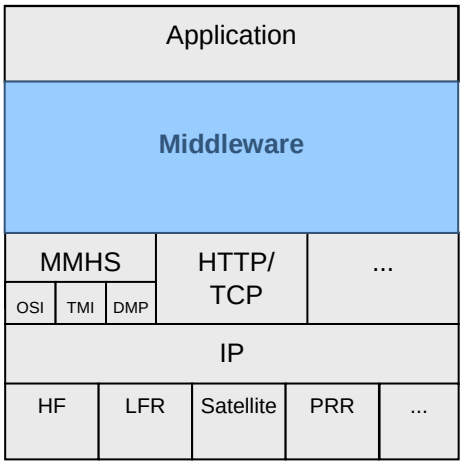

Figure 4.2: Middleware supporting multiple carriers with EoIP

In our preliminary experiments, we have devised a simple middleware which is capable of transporting data over messages in XOmail, a STANAG 4406 compliant message service developed by THALES. We focused on the STANAG 4406 Annex E tactical protocol profiles TMI-1 and TMI-4, which is most suited for use in disadvantaged grids. One of the things this middleware was used for was efficiency measurements of various compression techniques, as described in [19]. Rather than using a physical network, we used a link simulator as indicated in Figure 4.3. The experiments were all run using the default message priority level, i.e. routine.

We have configured XOmail to map different priority levels to the TOS field of the IP-header, and have implemented this mapping in our middleware also. It is interesting to note that XOmail is DiffServ compliant, but it consequently maps its priority levels to a corresponding yet higher level in DiffServ. The details of this priority mapping are shown in Appendix A. This simple, static middleware will form the foundation of future experiments with QoS in disadvantaged grids. In future experiments, the QoS needs will be handled by the middleware, where it will pay attention to application needs, and perform cross-layer mapping of priorities. This cross-layer mapping can in disadvantaged grids be done by using the mapping functions of XOmail, whereas for other carriers it would have to set the IP TOS field in a different manner. Furthermore, the middleware should monitor resources and adapt its mode of communications accordingly.

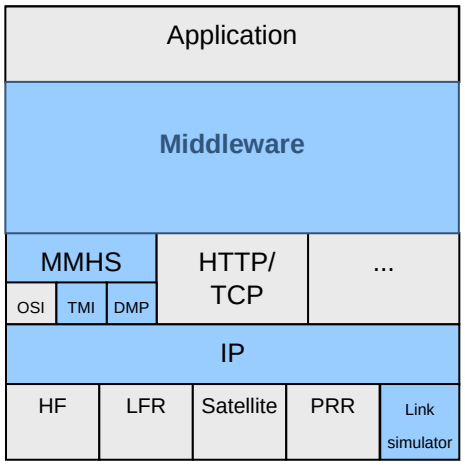

Figure 4.3: Middleware experiments using MMHS

5 Publish/subscribe using Web Services

In this chapter we present the current standardization efforts regarding Publish/subscribe for Web Services. Web Services are an enabling technology for NBD, and this makes new standards important to discuss. Furthermore, a publish/subscribe service is important for minimizing resource use in the system, and thereby help to achieve the necessary QoS.

Publish/subscribe is a well known communication pattern for event-driven, asynchronous communication. The pattern is particularly well suited in situations where information is produced with irregular intervals. A typical example of this is a sensor network, where information is produced each time a sensor makes an observation. Using a traditional pull-based pattern, the consumers of this information would have to poll the sensor at regular intervals, to check if there was new information available. By using the publish/subscribe pattern instead, the consumers subscribe to information from the sensors, and receive a notification each time new information is available. This means that the consumers receive the information the moment it is available instead of having to wait for the next polling, and the network traffic is reduced since polling is not necessary. Thus, the asynchronous nature of the publish/subscribe paradigm makes it a very important mode of communications in NBD.

At present there are two standardization efforts regarding publish/subscribe for Web Services: OASIS finished its Web Services Notification (WSN) standard [10] late in 2006, whereas W3C has produced a draft version of a similar framework called Web Services Eventing (WS-Eventing) [11] which may eventually become a standard. The WS-Eventing specification defines a baseline set of operations that allow Web Services to provide asynchronous notifications to interested parties. WS-Eventing is available as a public W3C draft [11], but is not yet a standard. It provides basic publish/subscribe functionality. WS-Eventing has similar features to that of WS-BaseNotification which we present below, so we will not go into the details of the draft. However, the techniques we present in later chapters should be possible to implement using either framework.

In this chapter we summarize the most important aspects from [10], [18], and [11]. We discuss event-based communications as seen from a military perspective. For examples of civilian applications of event-based SOA, see [18].

5.1 WS-Notification

WS-Notification is a Publish/Subscribe notification framework for Web Services. There are three parts to the specification: WS-BaseNotification, WS-BrokeredNotification and WS-Topics:

- **WS-BaseNotification** [15] defines standard message exchanges that allow one service to subscribe and unsubscribe to another, and to receive notification messages from that service. The WS-Eventing specification provides similar functionality to that of WS-BaseNotification, but they are not compatible with each other.

- **WS-BrokeredNotification** [16] defines the interface for notification intermediaries. A Notification Broker is an intermediary that decouples the publishers of notification messages from the providers of those messages; among other things, this allows publication of messages from entities that are not themselves Web Service providers.
- **WS-Topics** [17] defines an XML model to organize and categorize classes of events into "Topics," enabling users of WS-BaseNotification or WS-BrokeredNotification to specify the types of events in which they are interested.

The WSN specifications standardize the syntax and semantics of the message exchanges that establish and manage subscriptions and the message exchanges that distribute information to consumers. An information provider, known as a notification producer, that conforms to WSN can be subscribed to by any WSN-compliant subscriber.

WSN defines a set of terms, the most important of which we list below:

- **Situation**
  A situation is an occurrence (something has happened) that is noted by one party and is of interest to other parties.
- **Notification**
  WSN uses this term to refer to the one-way message that conveys information about a situation to other services. The association between a situation and the type of corresponding notification message is not necessarily one-to-one. It is possible that an application might associate several different notification message types with a given situation. This could be the case if there are multiple receivers and the aspects of a situation that are of interest to the receiver vary from receiver to receiver.
- **Publisher**
  A publisher is an entity that creates notification message instances (commonly known as events) based on a situation.
- **Notification Producer**
  A service that is responsible for sending notifications to the appropriate consumers. If the notification producer does not act as publisher, it is referred to as a notification broker. The notification producer is responsible for maintaining a list of interested consumers and arranging for notification messages to be sent to those receivers.
- **Notification Consumer**
  The counterpart of a notification producer is an entity that receives the notifications distributed by notification producers. The most common kind of consumer is a push consumer, which is able to receive notifications sent directly from the notification producer. WSN also supports pull consumers, which interact with the notification producer (or some intermediary) when they wish to receive information. The pull communications are performed with pre-defined pull-points at the service. Pull-points can for example be used by clients that join a network to synchronously "catch up" by fetching an aggregated report of previous events, for example the current common operational picture.

- **Subscription**
  An entity that represents the relationship between a notification consumer and a notification producer. It records the fact that the notification consumer is interested in some or all of the notifications that the notification producer can provide. In loosely coupled environments such as SOAs, it is often desirable to apply a finite lifetime to a subscription so as to avoid situations where consumers disappear or lose interest in a subscription without cancelling it. The subscription lifetime is a tradeoff between refresh rate and amount of "dead" subscriptions. Short-lasting subscriptions means that the clients frequently have to renew their subscriptions, but avoid the problem of "dead" subscriptions running for a long time, and vice versa.
- **Subscriber**
  Although subscriptions may be defined statically as part of a system design, event-driven architectures typically involve dynamic subscriptions. WSN uses the term subscriber to refer to an entity that requests creation of a subscription. Note that a subscriber may play the roles of both consumer and subscriber. WSN separates the two roles to allow third-party subscriptions.
- **Subscription Manager**
  The subscription manager is a service that manages requests to query, delete, or renew subscriptions. Each subscription manager is subordinate to the notification producer that owns the subscriptions in question. It is possible for a single service to take both the subscription manager and notification producer roles.

5.2 WS-BaseNotification

The WS-BaseNotification specification unifies the principles and concepts of SOA with those of event-based programming.

WS-BaseNotification provides the foundation for the WSN family of specifications. It defines the basic roles and message exchanges needed to express the notification pattern. The specification can be used on its own, or it can be used in combination with the WS-Topics and WS-BrokeredNotification specifications in more sophisticated scenarios. The specification defines the message exchanges between notification producer, notification consumer, subscriber, and subscription manager.

WS-BaseNotification does not specify the details of how notification message instances are created and does not define any interface between a publisher and the notification producer. In a tactical system one can use this functionality to integrate legacy systems into NBD. The legacy system can be considered a stand alone publisher, and be Web Services enabled by allowing it to publish through a WSN compliant notification producer.

The simplest form of a subscribe request message just contains an endpoint reference for a notification consumer. This form of request instructs the notification producer to send each and every notification that it produces to the notification consumer.

The subscribe request message can optionally contain one or more filter expressions. The filter expressions indicate the kind of notification that the consumer requires by restricting the kinds of notification that are to be sent for this subscription.

WS-BaseNotification defines the following three kinds of filter expressions:

- **Topic filters** provide a convenient way of categorizing kinds of notification. A topic is a concept used to categorize kinds of notification and their associated notification message types. A topic filter excludes all notifications which do not correspond to the specified topic or topics. Topics are standardized in WS-Topics, which defines topic trees and the use of namespaces. Further discussion of this standard is beyond the scope of this report, refer to [17] for the complete overview. In a military system we foresee extensive use of this standard. For example, one can perform filtering on tracks about only particular vessel or vehicle types, weather reports, etc.
- **Message filters** are Boolean expressions evaluated over the content of the notification message. This kind of filter can also be beneficial in military systems. For example, a squad on an operation needs only track information about the area it is operating in. Several squads operating out of the same local HQ should all get information relevant to them. Here a message filter using each squad's reported position can be employed to ensure that only tracks regarding their geographical location is sent.
- **Producer state filters** are based on some state of the notification producer itself. In order to use this kind of filter expression, the subscriber needs to know something about the properties of the notification producer.

5.3 WS-BrokeredNotification

The specification defines the concept of a notification broker as an intermediary Web Service that decouples publishers and notification producers. Thus, a notification broker is itself a Web Service, a notification producer and a notification consumer. It can be hosted remotely from both the publisher and the notification consumers, if required.

An implementation of a notification broker may provide additional added-value functions, for example logging notification messages or transforming topics or notification message content. Logging is important in military systems for security and auditing purposes. We discuss other added-value functions in the chapter on proxies, where we suggest further functionality to implement in a notification broker.

There are cases where it is expensive for a publisher to detect a situation or create a notification message instance. A problem with the simple publisher approach is that even when there are no relevant subscriptions, publishers still have to do both of these things. As an optimization, a broker may offer support for demand-based publishing. If the broker detects that there are no relevant subscriptions, it can pause its subscription with the publisher, resuming it again when it acquires a relevant subscription.

6 Proxy servers

A proxy is a unit which functions as an intermediary between two communicating parties [9]. Such an intermediate node can perform several different functions in a network; it can reduce overhead, increase security, increase availability and reduce access latency.

A proxy operates on the application layer; layer 7 in the OSI-model. This means that a proxy has access to more information than a router or a firewall that operates on lower layers of the OSI-model, enabling it to employ more advanced (content based) techniques for example when filtering. Thus, a proxy can improve QoS in many ways.

A proxy can be either transparent or non-transparent. A transparent proxy is invisible to the client, and can be used without changing existing client and server implementations. A non-transparent proxy, on the other hand, is not invisible to the client. A client needs to be configured properly to communicate via the non-transparent proxy.

Three of the most common proxy functions which improve QoS are:

- Connection management
- Caching
- Filtering and firewalling

Squid [8] is one example of a proxy that performs all these functions for web pages and is in widespread use on the Internet today. A notification broker may be regarded as a Web Services proxy. Apart from this, we are unaware of any proxy solutions that support Web Services. This chapter will start by discussing how proxy techniques currently employed for web pages on the Internet can be adapted for use with Web Services in NBD. Proxy solutions which support streaming media also exist. The latter is an area in which a lot of research has been performed in the later years, and it continues to be a "hot topic" in the academic research community since video streaming is gaining more and more popularity on the Internet. In a military scenario transmission of video may also play an important role, one example being the video feed sent by an UAV on a reconnaissance mission. At the end of the chapter we will present some video proxy techniques, and discuss their relevance for NBD.

6.1 Web Services

Web Services support two different communication paradigms; request/response and publish/subscribe. In a request/response service, the client will send an explicit request to a server, which in turn will process the request and respond with the requested data. In a publish/subscribe service, the client will subscribe to a service at a certain server, and the server will send new information to the client as soon as the data becomes available. Basically, request/response is traditional "pull" communications, whereas publish/subscribe is a combination of "push" and "pull" communications. We will now describe how proxies can be

employed to improve both “pull” and “push” communications in Web Services.

6.1.1 Connection management

Proxies can be used to handle many different types of network connections. One typical use for proxies is for connection sharing between many users.

The asynchronous nature of the publish/subscribe paradigm makes it a very important mode of communications in NBD. A notification service will, in its basic form, lead to the transmission of one notification to each of the consumers identified by the subscriptions per occurring event. This means that many copies of the same message will have to be sent over the same physical network connection if the consumers are located in the same area. This should be avoided if possible to save bandwidth, and multicast should be employed. Proxies can be used to introduce multicast communications even if the notification service itself does not support it, thereby saving resources and improving overall QoS.

The following example illustrates how a proxy can be used to increase the efficiency of publish/subscribe communications by reducing network load. A notification service has a server which handles subscriptions and the publication - dissemination of new information - to its subscribers. When a client wishes to subscribe to information from such a service, it needs to register its interest with the server. This is done by sending a “subscribe” message to the server, which then confirms that the message is received and that a subscription has been established. The illustrations below show how a proxy can be employed to optimize various aspects of the notification service. *Figure 6.1* illustrates a subscription establishment phase in which two clients are connecting through the same proxy, wanting to subscribe to the same service. First, client A sends a “subscribe” message to the server S. The proxy intercepts the message, noting that client A wants to subscribe to a service at S. This is the first time P sees a request for a subscription to this particular service, so it starts a subscription for this service at S. The server acknowledges that the subscription has been established, and the proxy forwards the acknowledgement to A. At this point S knows that whenever new information is available, it must send a notification to P. P, upon receiving such a notification, knows that it should forward it to A. Other clients may want to subscribe to the same service through P. In this example, B sends a “subscribe” message which is intercepted by P. P notices that it already has established a subscription to this particular service, adds B to its local subscription list, and sends an acknowledge message to B. Assuming that the clients in this example are squad members, and that the squad leader’s communications equipment has proxy functionality, then it is obvious that sending only one “subscribe” message over the low bandwidth link between squad leader and HQ avoids unnecessary messages and thus limits the use of the scarce bandwidth. By freeing communications resources in this way, other applications will have more available bandwidth for their network traffic. Thus, more information can be sent over the same network, increasing system efficiency and helping fulfil QoS demands.

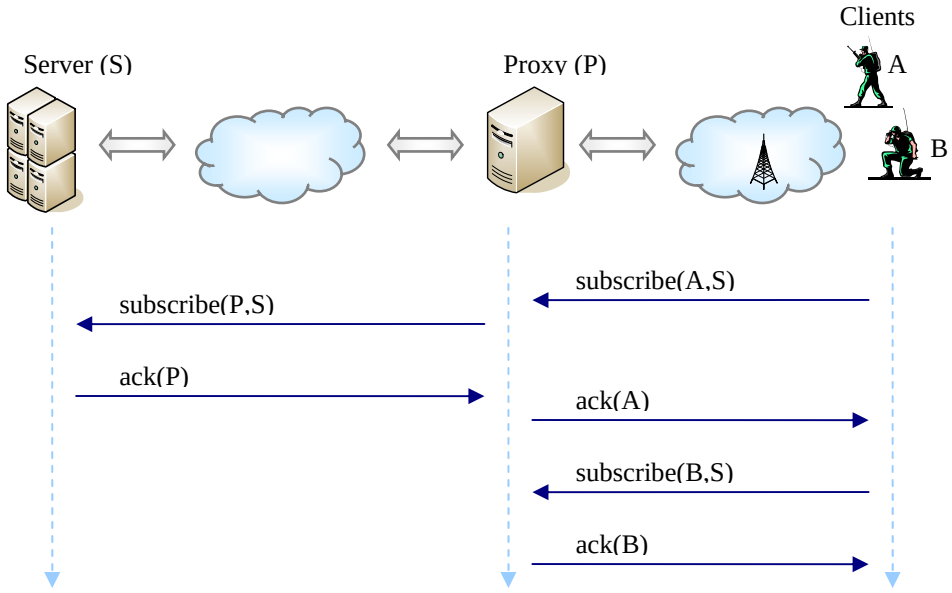

Figure 6.1: Subscription by proxy, for instance by WS-BrokeredNotification

When a subscription has been established further communications will be done asynchronously; the server publishes the content by sending a "notification" message to each and every subscriber. *Figure 6.2* illustrates this: S has new information for its subscribers (which in this case is only P), and sends a "notification" message. P has established this subscription on behalf of clients A and B, and will forward this message to both those clients. How P forwards the message will depend on the transmission medium between P and the clients. If the network does not support multicast, then P will have to send one "notification" message point-to-point to each client. Unicast distribution of messages to several clients on the same subnetwork should be avoided if possible to limit bandwidth usage. Multicast should be used instead if it is available, as shown in *Figure 6.3* where the proxy sends one multicast "notification" message to both the subscribing clients.

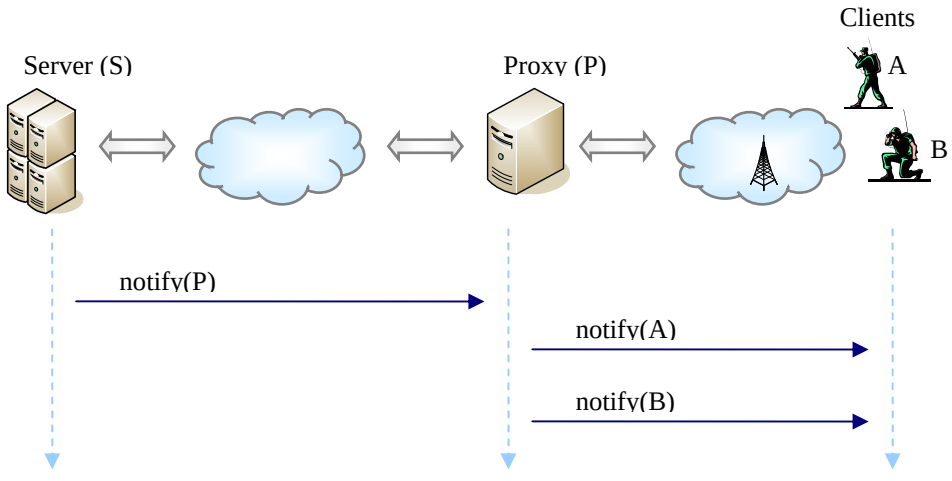

Figure 6.2: Notification via proxy, unicast

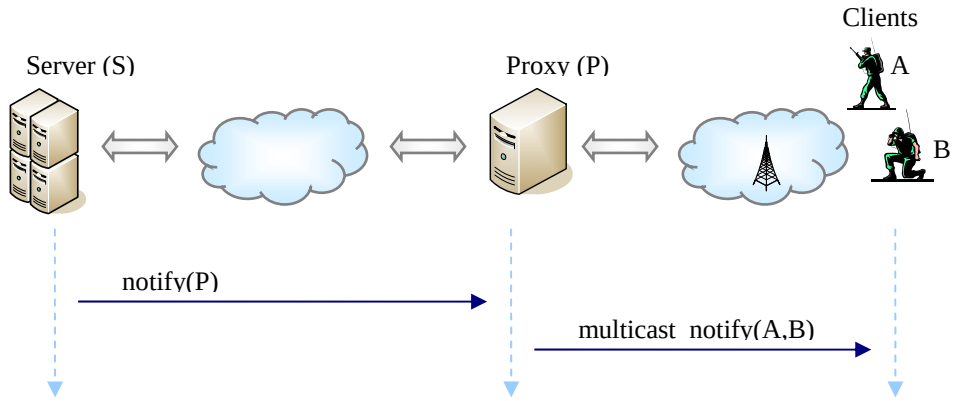

Figure 6.3: Notification via proxy, multicast

Using a proxy has both advantages and disadvantages: An important advantage is that bandwidth is saved between server and proxy. Furthermore, the proxy can adapt the mode of transmission to better utilize the transmission medium between itself and the clients by using multicast, for example. This can lead to a substantial reduction in bandwidth needs, since there will be only one message traversing the network per "notification" sent – no matter how many subscribers there are. This gives the system greater scalability. Further bandwidth savings can be achieved using filtering, which will be discussed below. All in all, proxies are important for QoS. Introducing a proxy also has some disadvantages. The proxy needs to keep state about subscriptions, and becomes a possible point of failure in the network. If a proxy crashes and loses its information, then it will not know what to do with incoming notifications, i.e. where to forward them. It is necessary to investigate various solutions for subscription renewal to overcome this problem.

6.1.2 Caching

Caching is a technique that is used to increase system responsiveness and decrease the response time for clients and reduce the load on the network and origin server, improving QoS. For example, it is in widespread use for accelerating web browsing on the Internet [12]. When someone accesses a web page, a local copy can be made in a proxy closer to the client. The next time someone requests the same page through that proxy, it can serve the client directly from its cache rather than having to fetch the page from the origin server. Caching is performed on the basis of observed client behaviour, and is a technique that is complementary to replication. Replication also leads to a copy of the original data being made, but it is controlled by the content owner. The origin server is responsible for making replicas and sending updates when content changes. Proxies are often used for caching, so called caching proxies. The use of caching proxies can be beneficial in disadvantaged grids, since resources are scarce. We will now discuss how the technique can be used with request/response driven Web Services. For simplicity, we will refer to a caching proxy as a proxy for the remainder of this report.

Figure 6.4 shows clients in a network, where a proxy is used to connect the sub-network the clients are on to the rest of the world. All communications to and from that network will pass

through the proxy. In the context of the scenario, the clients (A and B) can be seen as members of a squad. The proxy (P) is part of the squad leader's equipment since the squad leader can communicate with both the server (S), i.e. HQ and other squad members. If client A requests "object.x" from S, then this request will have to pass through P. P inspects the request, and discovers that "object.x" is not in its cache. The proxy forwards the request to S, which responds with the data. P can now store "object.x" in its local cache, and forward it to A. If another client later wishes to retrieve the same object, then it can be served directly from the cache, as is the case when B requests "object.x". This saves the time that would be spent retrieving the object from S, and leads to less communication between P and S, as well as less load on S since it will need to handle fewer requests.

Employing caching is especially useful if the server side network capacity is as shown in *Figure 6.4*, e.g. with the bottleneck between server and proxy. In such cases, all requests that P can handle without having to communicate with S will save bandwidth. Caching also increases the availability and robustness of the system; cached objects exist in the proxy even if the origin server becomes unavailable.

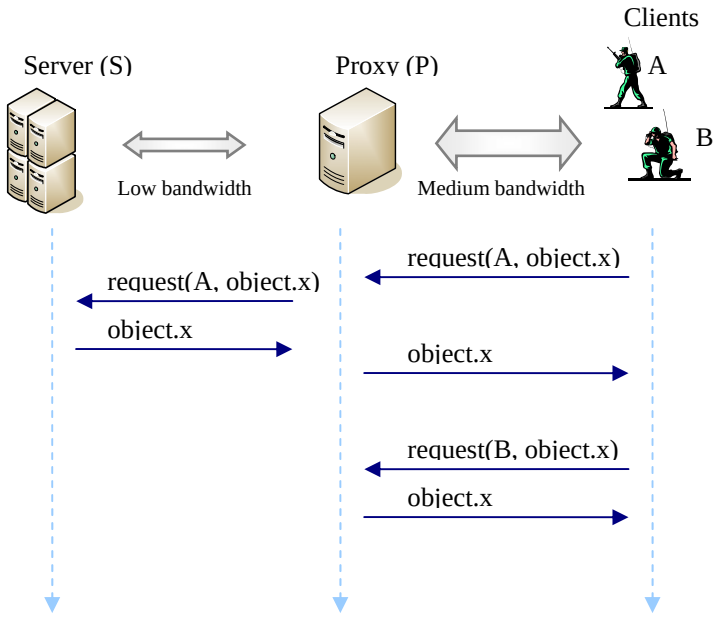

Figure 6.4: Caching

Implementing a caching proxy for Web Services is a challenging task. The proxy must have an understanding of the objects it caches in order to handle them in an appropriate manner. For example, if an object has a limited time that it remains valid, the proxy must know this. If a client requests the object after it has expired, then the proxy must fetch a new version of the object from the origin server. Caching is not efficient if objects change frequently. Furthermore, a proxy does not have unlimited storage space. If its cache becomes full, it will have to choose which objects to discard from the cache and which objects it should keep. Proxies often use simple statistical cache replacement algorithms [12], but there exist a lot of different algorithms [13] optimized for different conditions.

6.1.3 Filtering

In a disadvantaged grid it is important to avoid flooding the network with more information than it can handle. On the other hand, it is important that relevant information is sent to the receiver. To achieve this it is necessary to prioritize, filter and adapt information between networks. Filtering can be divided into three main types:

1. Filtering at the IP level, i.e. granting or denying access to or from specific IP addresses. This can be used in various security measures, but also to prioritize resource use.
2. Filtering of whole messages based on various criteria such as message priority, relevance and classification. Messages with high priority should be dispatched before messages with low priority.
3. Filtering of parts of messages based on the message content.

A proxy can perform all three types of filtering. Type 1 and, to a certain extent, type 2 filtering can be performed at the network layer by a router, whereas more advanced forms of filtering need application layer solutions. Type 3 filtering is content based, and requires the proxy to know not only the semantics of the content, but also the specific needs of each client. Therefore, it is important not only to have a standardized message format for the content, but also a standardized way to express QoS needs. In the following section we will discuss filtering in the context of multimedia proxies and quality adaptation.

6.2 Video

Proxy servers are important for a system's performance, and therefore also for achieving QoS. Different clients have different capabilities and resource requirements, something which demands media gateways and filtering.

An adaptive proxy has three main functions; caching, filtering and media gateway functionality. The proxy can perform media adaptation – so-called transcoding.

Traditional web proxies have two basic functions: Firewalling and caching. This basic functionality can be expanded by paying heed to the heterogeneity of the user mass; different needs, different network connections, and different hardware. In such a heterogeneous environment a proxy can assume a much more central and general role than usual by offering different user and terminal classes different classes of quality. It is important that the proxy understands which preferences and hardware resources a user has, and that it can handle different versions of the same information. For a Web Service, this can be filtering and adapting the amount of information (and thus the size of the XML document) which is sent to each user to meet the QoS demands. Correspondingly, a multimedia proxy will have to handle multiple versions of the same video, and protect user and network from overload by avoiding transferring something that exceeds the available resources. The proxy must match user needs with available resources, and perform filtering based on this. The goal is to give the user the quality needed, and not necessarily the best quality in all cases where it is possible.

Video can be divided into two categories; stored and live. A stored video is made available on a server, and can be retrieved when needed by a client. This means that the client contacts the server and gets the video from there, so-called Video-on-Demand (VoD). If more than one client wants the same video, then it will save resources to cache the video in a proxy in the clients' network. This reduces both origin server load and conserves bandwidth use between server and proxy. The proxy can cache the video in multiple versions with different quality. Quality variations can be handled efficiently by using video coded in layers, so-called layered video. At present, only MPEG-4 (ISO/IEC 14496) defines a standardized way to perform layering, and that part of the standard has not been implemented in any codec[2] [32]. There exist proprietary solutions used for research, such as QStream [31].

If a video has not been layered, then the proxy can perform transcoding on its own. Transcoding requires a lot of processing power, and this means that a proxy employing this technique can only handle a limited number of concurrent streams.

A media gateway will, in the same way as a caching proxy, inspect the data flow between server and client, but it may also modify the flow according to some pre-defined rules. The transcoding can be described by metadata (for example user preferences) or be hard coded. A media gateway will then transcode videos accordingly, giving them a certain set of properties such as a specific resolution, bit-rate, colour, etc, i.e. it will accommodate the format-oriented QoS parameters.

Bit-rate adaptation can accommodate bandwidth limitations. Resolution scaling also affects bandwidth needs, but the technique can also be used to adapt the video to circumvent known limitations in the client's hardware; processing power, memory and screen size.

Caching can be divided into two categories: Full caching and partial caching. Full caching is widely employed for caching Web objects, but is usually not the best alternative for videos. Videos are often big, and by employing full caching the cache will yield a low hit rate since only a few videos will be stored. By using partial caching one can store only a selected part of the video instead, for example only the beginning. Storing the beginning of the video is known as prefix caching [22], a technique which is used to reduce start-up latency. Such partial caching of the timeline of the video is also called temporal caching.

Caching in the quality domain can be performed in conjunction with employing a layered codec. A layered codec reduces the problem of quality adaptation to just choosing which layers to cache and send to the clients.

---

[2] A *codec* performs media-data *co*mpression and *dec*ompression. When a track is encoded, it is converted to a compressed format suitable for storage or transmission; when it is decoded it is converted to a non-compressed (raw) format suitable for presentation. Each codec has certain input formats that it can handle and certain output formats that it can generate. In some situations, a series of codecs might be used to convert from one format to another.

Most codecs do not support layered coding even if this is a prerequisite for quick and effective adaptation (the alternative being resource demanding transcoding). Adaptation of non-layered videos leads to problems. The first problem is that it demands a lot of processing power due to the resource demanding decoding and coding operations. This limits the scalability of the system. The second problem is the reduced hit-rate when caching. One video can be stored in several different versions depending on which transcoding has been performed. The cache can contain the same video with many different levels of quality, limiting the number of different videos that can be stored. The sum of the file sizes of the different transcoded videos will be greater than if one codes a video once with a layered format (assuming that the compression used is the same). This means that one should keep an eye on layered video research and standardization efforts, since the use of such formats will demand less CPU power and storage space.

7 Summary

QoS management is an important aspect in the realisation of the NBD vision. The goal of this report has therefore been to give a better understanding of QoS and the mechanisms required for managing QoS in a distributed system.

We first provided a detailed discussion around QoS, presenting different categories and semantics of QoS, before discussing how QoS is dealt with in NNEC FS. Next, we described how cross-layer signalling is necessary to achieve end-to-end QoS, and how DiffServ is an important part of this.

In a distributed system, middleware is probably the most important mechanism for enabling QoS management, since it is a pervasive element within the system. We have therefore given a short overview of existing approaches to middleware, and discussed the shortcomings of these approaches, with respect to both QoS and disadvantaged grids.

Since NATO currently focuses on using Web Services, we have provided a short discussion around the relationship between traditional middleware and Web Services middleware, and how this technology can be used in tactical communication networks. We have also given an overview of how the publish/subscribe mechanism is realized with Web Services, since this is a very relevant mechanism in an NBD context.

The use of proxy servers can turn out to be very useful, in particular in disadvantaged grids, and we have therefore given a short introduction to the principle of such servers, and then discussed in more detail how they can be used together with Web Services in disadvantaged grids. We also gave an overview of how proxy servers can be used for QoS management in connection with video transfer.

This report represents part of the outcome of the FFI project 898 “NBF Beslutningsstøtte”. At the same time, it presents areas that must be investigated further, in order to realize the NBD vision. Therefore, this report will also function as a starting point for the FFI project 1086 “Sikker gjennomgående SOA”, in which these areas will be an important part.

References

[1] Griwodz, C., Liepert, M., Saddik, A. E., On, G., Zink, M., and Steinmetz, R. 2001. Perceived Consistency. In *Proceedings of the ACS/IEEE international Conference on Computer Systems and Applications* (June 25 - 29, 2001). AICCSA. IEEE Computer Society, Washington, DC, 260.

[2] Ghinea, G., Thomas, J. P., and Fish, R. S. 1999. Multimedia, network protocols and users—bridging the gap. In *Proceedings of the Seventh ACM international Conference on Multimedia (Part 1)* (Orlando, Florida, United States, October 30 - November 05, 1999). MULTIMEDIA '99. ACM Press, New York, NY, 473-476.

[3] On, G., Schmitt, J., and Steinmetz, R., 2002. On Availability QoS for Replicated Multimedia Service and Content. In *LNCS 2515 (IDMS-PROMS'02)*, pp. 313326, Portugal, Nov. 2002.

[4] Quality of Service for Web Services in NEC (in Norwegian)
Johnsen, F.T. and Hafsøe, T.
FFI/NOTAT-2006/02580

[5] TACOMS POST 2000
http://www.tacomspost2000.org/

[6] Interoperable Networks for Secure Communications, Final Report Phase I
http://insc.nodeca.mil.no/ifs/files/public/INSC%2DI/Task1/T1%2DD011_Final_report.doc

[7] Computer Networks, 4th edition.
Tanenbaum, Andrew S.
2003 Pearson Education, Prentice Hall PTR. ISBN 0-13-038488-7

[8] Squid Web Proxy Cache
http://www.squid-cache.org/

[9] Proxy Servers Tutorial – About Proxy Servers
http://compnetworking.about.com/cs/proxyservers/a/proxyservers.htm

[10] OASIS Web Services Notification (WSN) TC
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

[11] W3C Web Services Eventing (WS-Eventing) public draft release
http://www.w3.org/Submission/WS-Eventing/

[12] WEB Caching and Replication
Rabinovich and Spatscheck, Addison Wesley 2002, ISBN 0-201-61570-3

[13] Christine Küfner, Carsten Griwodz,
"Möglichkeiten für den Einsatz von Lastverteilungsstrategien verteilter Systeme in der Videoverteilung" (in German), KOM TR-1998-07, KOM TU Darmstadt, November 1998

[14] Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju. 2004. Web Services Concepts, Architectures and Applications, Springer

[15] WS-BaseNotification 1.3 OASIS Standard, approved October 1st 2006
http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

[16] WS-BrokeredNotification 1.3 OASIS Standard, approved October 1st 2006
http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf

[17] WS-Topics 1.3 OASIS Standard, approved October 1st 2006
http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf
[18] P. Niblett and S. Graham, Events and service-oriented architecture: The OASIS Web Services Notification Specifications, IBM Systems Journal, volume 44, no 4, 2005
[19] Web Services in networks with limited data rate (in Norwegian)
Dinko Hadzic, Trude Hafsøe, Frank T. Johnsen, Ketil Lund, Kjell Rose
FFI/RAPPORT-2006/03886
[20] Efficient XML Interchange Working Group
http://www.w3.org/XML/EXI/
[21] Ram Ramanathan and Jason Redi, A brief overview of ad hoc networks. Challenges and directions. IEEE Communications Magazine, 50th Anniversary Commemorative Issue, May 2002 (Invited commentary)
[22] Subhabrata Sen, Jennifer Rexford, Donald F. Towsley: Proxy Prefix Caching for Multimedia Streams. INFOCOM 1999: 1310-1319
[23] En oppsummering av NBF tenketank nr 5, 24-25 januar 2005 (in Norwegian), Bård Reitan, Geir Enemo
FFI/NOTAT-2005/00504
[24] Anton Leere, FFI. Personal communication, autumn 2006.
[25] Lorns Bakstad, FFI. Fredagsforum 8. desember 2006. Presentation (in Norwegian) titled "UAV".
[26] Bakken, D. E., 2001, Middleware. *Chapter in Encyclopedia of Distributed Computing*, Urban, J. and Dasgupta, P. (eds.), Kluwer Academic Press.
[27] Sun Micro Systems, Java 2 platform enterprise edition specification, v1.4, 2003
[28] Vogel, A., Kerherve, B., von Bockmann, G., Gecsei, J., Distributed Multimedia and QoS: A Survey, IEEE Multimedia, Vol. 2, No. 2, 1995, pp. 10-19
[29] Danthine, A., Bonaventure, O., From Best Effort to Enhanced QoS, in International Workshop on Architecture and Protocols for High-Speed Networks, Schloss Dagstuhl, Germany, August/September 1993, pp. 179-201
[30] Özsoyoglu, G., Snodgrass, R., T, Temporal and Real-Time Databases: A Survey, IEEE Transactions on Knowledge and Data Engineering, Vol. 7, No. 4, 1995, pp. 513-532
[31] QStream home page
http://qstream.org/
[32] Johnsen, F.T., Electronic Imaging 2007 / Multimedia Computing and Networking – travel report, FFI-reiserapport 2007/00687

Appendix A Network level QoS support in XOmail

XOmail is the STANAG 4406 compliant military message handling system (MMHS) implementation we use for our experiments. It can map message priority to the TOS field (see Figure A.1) in the IPv4 header, the values for each MMHS priority are shown in Table A.1. A detailed breakdown of the TOS field is given in Table A.2. DiffServ redefines the TOS field into a DSCP and an unused part as shown in Table A.3. The DSCP defines IP packet precedence and priority which DiffServ enabled routers use to classify packets. An overview of the different DSCPs and traffic classes is shown in Table A.4.

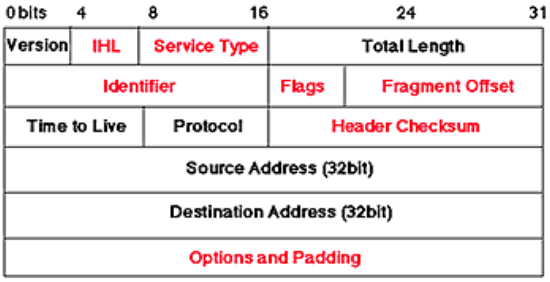

Figure A.1: IPv4 header, the field "Service Type" is also known as the Type-of-Service (TOS) field.

MMHS priority	HEX header value	DEC header value
routine	0x20	32
priority	0x40	64
immediate	0x60	96
flash	0x80	128
override	0xA0	160

Table A.1: MMHS priority values

bit 0	1	2	3	4	5	6	7
Precedence			**Delay**	**Throughput**	**Reliability**	**Cost**	**MBZ**
000 (0) - routine 001 (1) - priority 010 (2) - immediate 011 (3) - flash 100 (4) - flash override 101 (5) - critical 110 (6) - Internetwork Control 111 (7) - Network Control			0 - normal 1 - low	0 - normal 1 - high	0 - normal 1 - high	0 - normal 1 - low	checking bit (Must Be Zero)

Table A.2: The TOS field is 8 bits in the IP header, where the first three define priority, followed by one bit for delay, throughput, reliability, cost and a control bit.

bit 0	1	2	3	4	5	6	7
Differentiated Services Codepoint (DSCP) Class selection is in the first 3 bits, and maps directly to the IP Precedence bits from the ToS table above.						**CU** Currently unused, best kept at 00 for backward compatibility with ToS.	

Table A.3: The DiffServ field

Name	Codepoint (CU)	DiffServ (decimal)	RFC #	Notes
CS0	000 000 (00)	0	2474	class 0, default (routine)
CS1	001 000 (00)	32	2474	class 1 - similar to the ToS Precedence (priority)
AF11	001 010 (00)	40	2597	AF (Assured Forwarding) class 1 - low drop precedence
AF12	001 100 (00)	48	2597	AF class 1 - medium drop precedence
AF13	001 110 (00)	56	2597	AF class 1 - high drop precedence
CS2	010 000 (00)	64	2474	class 2 - similar to the ToS Precedence (immediate)
AF21	010 010 (00)	72	2597	AF class 2 - low drop precedence
AF22	010 100 (00)	80	2597	AF class 2 - medium drop precedence
AF23	010 110 (00)	88	2597	AF class 2 - high drop precedence
CS3	011 000 (00)	96	2474	class 3 - similar to the ToS Precedence (flash)
AF31	011 010 (00)	104	2597	AF class 3 - low drop precedence
AF32	011 100 (00)	112	2597	AF class 3 - medium drop precedence
AF33	011 110 (00)	120	2597	AF class 3 - high drop precedence
CS4	100 000 (00)	128	2474	class 4 - similar to the ToS Precedence (flash override)
AF41	100 010 (00)	136	2597	AF class 4 - low drop precedence
AF42	100 100 (00)	144	2597	AF class 4 - medium drop precedence
AF43	100 110 (00)	152	2597	AF class 4 - high drop precedence
CS5	101 000 (00)	160	2474	class 5 - similar to the ToS Precedence (critical)
EF PHB	101 110 (00)	184	2598	Expedited Forwarding (recommended for video/audio - high priority, higher drop probability)
CS6	110 000 (00)	192	2474	class 6 - similar to the ToS Precedence (Internetwork Control)
CS7	111 000 (00)	224	2474	class 7 - similar to the ToS Precedence (Network Control)

Table A.4: Differentiated Services Code Point

If we compare Table A.1 with Table A.4, we see that XOmail consequently chooses one priority class higher in DiffServ than the corresponding message priority class in MMHS. Message "deferred" is DiffServ "routine", message "routine" is DiffServ "priority", and so on. XOmail only changes the first three bits of the TOS/DiffServ codepoint.