Requirements of a Middleware for Managing a large heterogeneous programmable Network

Andrew Hughes
Department of Computing Science
University College London
Gower Street, London, WC1E 6BT
a.hughes@cs.ucl.ac.uk

Abstract[‡]

Programmable networking is an increasingly popular area of research in both industry and academia. Although most programmable network research projects seem to focus on the router architecture rather than on issues relating to the management of programmable networks, there are numerous research groups (discussed in [55]) that have incorporated management middleware into the programmable network router software. However, none seem to be concerned with the effective management of a large heterogeneous programmable network. The requirements of such a middleware are outlined in this paper. There are a number of fundamental middleware principals that are addressed in this paper: these include management paradigms, configuration delivery, scalability and transactions. Security, fault tolerance and usability are also discussed; although these are not essential parts of the middleware, they must be addressed if the programmable network management middleware is to be accepted by industry and adopted by other research projects.

1 Introduction

The Internet started out as a research project, funded by the Defence Advanced Research Projects Agency, called ARPANET. The project focused on best-effort routing mechanisms that were designed and implemented in the hope that it would still be possible for military computers to communicate if nuclear war broke out. Since then, the Internet has been through a number of evolutionary cycles; it now supports a multi-billion pound industry that mainly revolves around e-commerce, media delivery and service provision. Although the way in which the Internet is used has drastically changed since its early days, the fundamental technology has changed very little. For the vast majority of Internet communications, data is transmitted in a best-effort manner. Communications generally have no relationship to business models: data that generates large revenues is not treated differently to those that generate little or no revenue. Since the Internet is becoming more and more congested, e-businesses are keen to control internal network congestion and provide services in the most profitable way possible. The highly restricted functionality of most 'off the shelf' routers does not allow packets to be routed in a highly customised way defined by the router's owner; Internet routing technology needs to be evolved to cater for the new requirements. Programmable networking technology is likely to be accepted by industry as the solution to this problem. By programming network routers, the process of routing data can be controlled in a highly flexible and customisable manner; this allows Internet corporations to control network congestion according to a business model.

Programmable networking is a relatively new area of research, the technology is therefore in its infancy. The main focus of research in this area seems to be the software architecture of the programmable network routers that constitute the programmable network. There are many approaches to network programmability, including Active Networks [52] and OpenSig [48] projects, however all take the view that the software that controls the routing process can be modified. This project does not intend to focus on a particular programmable network paradigm; rather, the aim is to research a middleware that can be used to manage a wide range of programmable network architectures.

According to [63]: middleware is the term usually given to the software layer that abstracts communications concepts from the designer of a distributed application; it is located between the operating system and the application. The concept of middleware is not a new: it is the focus of a very active research area and also has a strong presence in many modern day software engineering projects. There are numerous middleware types (some of are discussed later), all of which have a diverse range of applications. From the above definition, it is fairly clear that the process of managing a network consisting of many programmable routers can benefit from middleware: it is possible to add functionality without complicating the design of the system.

Many research groups seem to agree that, due to the increased complexity of programmable network routers (with respect to current 'off the shelf' routers), management middleware is required to allow the efficient and correct configuration of the nodes. However, programmable network management middleware is generally not a high research priority. Of those projects that do approach programmable network management issues, it is generally accepted that programmable network routers will be configured individually rather than in groups. This paper takes the view that this approach is not acceptable: the

[‡] This research was supported by BTExact Technologies [1]

programmable network management middleware should be scalable.

This project attempts to separate the management middleware from the programmable router architecture, thus allowing the middleware to be deployed into a diverse range of programmable network architectures and also enabling the management of a set of programmable routers with differing software architectures. We refer to this as an homogenous programmable network. The programmable routers, the administrators, and nodes that provide middleware mechanisms are collectively termed as a 'community'. We consider a node to be any location that the middleware can communicate with: instances of programmable network routers are nodes whereas network interfaces and computers are not.

This paper outlines the requirements of a middleware intended to simplify the task of managing a large programmable network; the rationales for the requirements are also presented. It must be noted that this paper is not a design document: the architecture of the middleware and details on the implementation of the mechanisms outlined in this paper will be addressed in the design phase of this project.

The paper is structured as follows: Section 2 discusses the fundamental mechanisms required to manage a community, Section 3 presents arguments in support of a transactional middleware, Section 4 approaches security issues, and Section 5 considers the usability requirements. Related work is presented in Section 6, conclusions are drawn and future work in outlined in Section 7.

2 Middleware Fundamentals

In this section, the fundamental mechanisms and principles of the middleware are discussed. This is done in five parts. We first discuss ways in which a community can be managed, and what support there should be in the middleware for these management techniques. We then illustrate the issues regarding communication between community members, and after that describe how community members discover the address of other nodes. We outline the support required from the network and discuss scalability requirements before finally outlining the four aspects of transactions.

2.1 Management

Management is the process whereby an administrator configures a node. The term 'administrator' is used to describe a process that is either performing management operations or capable of performing management operations in response to some event. A node which has performed management operations, but has since revoked its compliance with the definition of an administrator, should not be referred to as an administrator. Similarly, nodes which will perform operations in the future, but do not yet comply with the definition, are not considered as administrators. Since all administrators must comply with the middleware requirements set out in this paper, administrators are community members.

The programmable network can be managed in a number of ways (defined by its designer): it could be autonomous (such as in Android [47]) or done by a human administrator who interacts with some (perhaps graphical) user interface (such as in Promile [45]). Often, management decisions will be based on information retrieved from community members. Administrators modify the configuration of community members (also known as 'target nodes') by sending Configuration Messages (CMs) containing management instructions. The messages sent from a target node to an administrator are referred to as 'Notification Messages (NMs)', these messages can contain state information and event notifications. The format of the data contained in both CMs and NMs is not specified by the middleware; the designer of the programmable network is responsible for defining the format of the configurations and event notifications. The configuration messages sent to programmable network nodes may, for example, be formatted in XML [2] according to a Schema [3]; alternatively, the management instructions could be embedded in an executable program known as a delegate [4] (this management paradigm is similar to that presented in [5]).

Since the middleware is not aware of the data contained in CM and NMs, it cannot analyse or process the management instructions – that functionality must be provided by the user of the middleware. Model checking and consistency checking should be incorporated into the software that uses the middleware. A consistency checking and/or a model-checking module could be built into the graphical interface to the administrator software. Consistency checking could benefit from work done on the Xlinkit project [6], model checking could benefit from work done in the SPIN project [7].

The configuration of community nodes can be initiated by either an administrator or a target node, these paradigms are respectively termed 'Administrator Initiated Management' (AIM) and 'Target Initiated Management' (TIM). The middleware should support both of these paradigms. Examples both AIM and TIM are outlined below.

An example that provides a rationale for an AIM, consider an Internet Service Provider (ISP) who owns a network connected to the Internet by the ISP's own programmable network router through a non-proprietary link provided by British Telecom (BT). When the network was first constructed, BT charged a fixed monthly rate for unlimited bandwidth over the link to the Internet; however, BT has since changed their charging strategy in an attempt to reduce internal

congestion. The new charging strategy is such that if the outbound bandwidth usage exceeds 100Mbps, the monthly charge will be quadrupled. The ISP cannot afford to pay the extra charges, so decides to configure the programmable network router to ensure the (burst) bandwidth usage remains under the limit. The configuration begins with a human using a graphical interface to the administrator. The administrator sends a CM to the programmable network router instructing that a 'shaper' is used to ensure that that no more than 100Mbps are sent across the BT link. The programmable network router receives the CM and successfully processes it and commits the changes. The router then sends a NM to the administrator indicating that the configuration was a success. Once the administrator receives the NM, functionality provided by the developers of the programmable network graphical interface notify the human that the task was completed successfully.

The main role of TIM is to allow the community to be managed in an event driven manner. For a usage example of TIM, consider an extension to the scenario described in the previous example. If the programmable network router is configured to notify a community administrator if a link failure occurs, the AIM paradigm is not appropriate because an administrator would have to continuously poll the router to see if any problems have occurred. TIM allows target nodes to initiate communications with administrators; this paradigm is better for event driven management than AIM. The programmable network router, once configured, will monitor the state of the link (the way in which this is done is beyond the scope of this paper); if the link fails, the router initiates communications by sending an NM to an administrator. The way in which the administrator reacts to this event notification message is defined by the designer of the programmable network management system; in this example, the administrator is programmed to notify a human of the failure through the graphical interface. It should be noted that, like in this example, it is not a requirement that messages must be acknowledged on reception.

Both of the above examples omit details of node discovery. In AIM, administrators must be able to discover target nodes; TIM requires that community members can locate administrators. Location discovery is discussed in the later in this section.

2.2 Configuration delivery

Any community member can become an administrator at any time; therefore, in order for a community to be effectively managed, it must be possible for communication to take place between any two community members. As described in the previous subsection, communication is done using messages. In order for the middleware to be usable in heterogeneous programmable networks, the messages should be portable. This subsection describes the requirements relating to the delivery of these messages. The reader should note that although the term 'message' is used to describe the how data is transferred between nodes; the middleware may not be based on message oriented middleware concepts. The way in which data is transferred between nodes will be addressed in the design phase of this project. No assumptions have been made about the middleware's architecture; it may be based on message-oriented middleware such as JMS [8] and DyNet [9], mobile agent middleware such as Telescript [10] and Aglets [11], object oriented middleware such as CORBA [12] and RMI [13], or some other technology.

In this subsection, we first outline the requirements relating to two message routing models and then go on to discuss community membership and quality of service issues.

2.2.1 Point to point delivery

In order for an administrator to use the middleware to manage community members, the middleware must provide a delivery mechanism. The middleware must be able to route a message to the appropriate place so the destination must be embedded in the message. Since the sender of a message will often require a reply to be sent, the message source should also be included in the message. Clearly, every community member must have a unique identifier. It cannot be guaranteed that the IP address of every community member unique (the reasons for this are given in the 'network structure' subsection) so they should not be used alone to identify nodes, a better solution is to use character strings to differentiate between nodes.

The programmable network's designer should be able to assume that sent messages will reach the intended destination (this is discussed further in the 'fault tolerance' section). If it is required that the receiver is to notify the sending that a message has been received, the functionality must be provided by the programmable network management software. Messages should also comply with some structure definition so that it is possible to discover when invalid messages are received. If this happens, an error notification should be returned to the sender that sent the invalid message.

The middleware should support the delivery of messages to one, many or all of the nodes in the community. The delivery paradigms are respectively termed unicast, multicast and broadcast. Most multicast protocols (including the IP multicast protocol [14] supported in most 'off the shelf' routers) take the view that a small number of nodes will send multicast messages; however the management middleware requires that any community member can send multicast messages; this is termed as many-to-many multicast. Below are examples of how multicast and broadcast can be used in a programmable network system.

Consider a corporate network consisting of one thousand routers, of which one hundred routers are also connected to external non-proprietary networks; 'edge router' is the term used to describe this kind of router. Due to a public security announcement, the company decides to disallow all FTP [15] connections originating from outside the corporate network. This can be done by (the administrator) constructing a multicast message – addressed to all of the edge routers – which installs the appropriate 'dropper' rule.

To understand the purpose of broadcast messages, once again consider the corporate network described in the above scenario. A module vendor has significantly optimised a programmable network module and distributed a patch to its customers. The network owner decides that all of the routers in the network are to be updated with this patch. By using a broadcast message, an administrator can send a single configuration message to every community member. Obviously, this is far more straightforward than specifying the addresses of all nodes (the discovery of all the nodes may be a difficult task in itself).

It is worth being aware that the tasks above could be carried out by administrator sequentially updating each node individually with unicast messages; however, the multicast technique is far more efficient than multiple unicasts. If the administrator performs all of the updates simultaneously, it is highly likely that it will take significantly longer than the multicast method: the multicast message need only send a single message, whereas the concurrent method sends a message to each and every target node. Since it is unlikely that there will be completely unlimited bandwidth, it will take longer to deliver the configurations to every node using multicasts because of the increased bandwidth demand.

It is likely that the programmable network will require that nodes are grouped according to their functionality. Since the functionality of community members is not in the scope of the middleware, there is no need for it to provide grouping capability at this level. It is worth noting that the middleware's multicast and publish-subscribe (discussed in the next subsection) mechanisms can be used to send messages to all the nodes in - what the application may consider to be - a group.

2.2.2 Publish-subscribe delivery

The middleware should be capable of publish-subscribe message delivery. In this paradigm, messages are delivered to only those nodes that explicitly express an interest in receiving notifications. A node registers interest by subscribing to a 'topic'. To send a message to all community members that are subscribed to a topic, messages are 'published' to that particular topic.

Compared to multicast, far less knowledge of the community is required by the administrator in the publish-subscribe model. Depending on the programmable network, this may be an advantage or a disadvantage. On the positive side: by delivering messages to nodes that have registered interest in a topic, the administrator does not have to be aware of the receiving node's existence, and hence the addresses of all the locations to which a multicast message is to be sent. It should be noted that in order to subscribe to a topic, nodes must be capable of discovering the available topics; this will be addressed in the design phase of this project. On the negative side: unlike the multicast model, an administrator using the publish-subscribe model cannot be sure that a message will be received by any specific nodes.

As an example of how publish-subscribe could be used in a programmable network, consider the multicast example given in the previous subsection: in order to send updates to all of the edge routers, an administrator must have previously discovered the locations of all nodes which comply with the 'edge router' definition. In the publish-subscribe model, this foreknowledge is not necessary. Providing that all of the edge routers have registered an interest in the topic 'edge router updates' (for example), to configure all target nodes the administrator need only publish a message to the given topic.

2.2.3 Community membership

It is not realistic to make the assumption that the community membership is static. Nodes may join and leave a community for many different reasons, all of which are due to either faults or decisions made by the programmable network management system's designer. In a programmable network, nodes may be added to the community when new routers are installed or removed when there are redundant or faulty routers. The community membership is dynamic in that the middleware must allow nodes to join and leave the network at any time. When a node joins or leaves, the community membership is said to have changed. The middleware cannot make any assumptions regarding the frequency at which the community membership changes; therefore, the middleware must be capable of functioning correctly for a range of frequencies from occasional to frequent community membership changes.

In order for a node to join the community, it must be running a middleware which both complies with the requirements set out in this paper and is compatible with the middleware running in the community it wishes to join. Compatibility problems may arise due differing designs derived from the requirements (outlined in this paper) or because the middleware implementations are at differing points in the software lifecycle (i.e. they have incompatible version numbers).

Once a node joins a community, it may be the case that – depending on the design - some configuration has to be performed on the joining node's middleware. For example, if a node has to rejoin the community after a

failure, the previous settings should be applied. This configuration should be autonomous and transparent to the developer of the programmable network. However, since the middleware is not aware of the programmable network's functionality, it cannot be responsible for automatically performing programmable network specific configuration when a node joins a community. If this behaviour is required, the node configuration must be outlined by the user of the middleware.

2.2.4 Quality of service

Since the middleware is likely to be used to manage live systems, messages must be delivered with high quality-of-service: the time in which a message is delivered should be as low as possible. A significant delay between the sending and receiving of a message may be unacceptable. For example, if a node is checked to see if it is alive (a 'ping' message), the node will be considered to have failed if it does not respond in a reasonable amount of time. Since few systems require that configuration updates must meet hard real-time deadlines, the middleware will not guarantee that deadlines can be met.

As previously discussed, no assumptions can be made about the size of a message. Since the size of a message and the speed of the connections between community members are the main factors which affect the time required to deliver a message, when the speed of the connections between community members is low, it may take a long time to send a large message. If the middleware does not allow the nodes to send and receive multiple messages at the same time, the delivery of the large message must complete before any more (probably smaller) management message can be sent, thus adversely affecting the community's quality-of-service.

2.2.5 Location discovery

As previously discussed, the management of a community relies on the communications between community members. In the AIM paradigm, administrators must be capable of locating community members that possess certain properties; for example, a programmable network router running on the Linux [39] platform. The TIM paradigm (discussed in the 'management' subsection) requires that community members can locate administrators; for example, a programmable network router may have to send a link failure notification to an administrator which can react to this event. Clearly, there is a need for a mechanism which can, given some properties, return the unique identifier of nodes that exhibits them.

It may be the case that an administrator has a fixed location, however the middleware cannot assume this. For example, it is not unlikely that a corporate network administrator - which includes a graphical user interface – will be run on various community members depending on the location of the human network manager. Since it is the community members that have identifiers and not the administrator processes, it is not possible to send messages to administrators if the identifier of the host community member is not known.

Services such as JNDI [16], JINI [17] and LDAP [18] provide functionality that is useful for the discovery of objects' locations; however, these mechanisms would have to be extended to provide all of the required functionality for the community member discovery outlined in this subsection.

2.3 Network architecture

The middleware should allow the community to be heterogeneous: it should be possible to configure software, written in various languages, residing on a number of different operating systems. Cross-platform configuration should be transparent to the user of the middleware. If it is required that the programmable network is capable of discovering information about the platform, the functionality would have to be incorporated by the programmable network's designer. In some programmable network architectures, platform based configurations may be necessary as the modules that implement functionality are implementation specific.

The middleware should run over both proprietary and non-proprietary networks and be capable of crossing administrative domain boundaries. The middleware will be built on Internet Protocol (IP) [19] because this network protocol is widely used and mature in its development. The middleware must function correctly on asymmetrical networks, and when the data routes between community members is unreliable. If the community is deployed over a non-proprietary network, it cannot be guaranteed that community members are not separated by third-party hardware (such as routers). Care should be taken to ensure that these intermediate routers are not able to modify/corrupt en-route data.

Networks may be configured (either manually by a human administrator or automatically by some routing protocol such as OSPF [20] and BGP [21) such that the route that an IP packet takes is not the quickest route. Depending on the design of the middleware, this inefficiency may mean that a community consisting of a large number of nodes cannot provide an acceptable quality-of-service. The design phase of this project should address this issue.

Multiple nodes in a community may have the same IP address. There are three reasons why this could be the case: there may be multiple community members resident on a network interface; multiple connections connect to the community through a tunnel (for example SOCKS [22] or a NAT [23] firewall); and also because, when the community spans private networks, it cannot be guaranteed that the address ranges do not interfere (for example, multiple private networks may use the IP addresses reserved for use in private networks by The Internet Corporation for Assigned Names and Numbers [24]). The latter is

likely to occur when the programmable network crosses management domain boundaries.

2.4 Scalability

The middleware must be scaleable in that the quality-of-service in the community should be acceptable when the community membership ranges from few, to many nodes. The number of nodes that the middleware can configure should be significantly larger than that which a human systems administrator could configure manually in a reasonable amount of time.

There are some situations where it is viable for a system administrator to manually configure a community. For example, a medium sized Internet Service Provider offering a 'dial-up' service will generally own a small number of routers which are configured by hand. When a community consists of a large number of nodes, it is no longer feasible to use the manual management approach. This is certainly the case for a service provider that offers television, telecommunications and broadband Internet access. Companies such as such as NTL [25], Telewest [26] and British Telecom [27] (who provide cable TV and Internet) have networks that consist of thousands of routers. It is therefore realistic to aim for a middleware scalability of up to 10000 nodes.

2.5 Transactions

The configuration of a number of community members may have to be performed with transactional properties. There are four transactional properties: atomicity, consistency, isolation and durability; these are known as the ACID properties [28]. The atomic property ensures that a sequence of instructions has an "all-or-nothing" property. If faults occur, the system can role back to a previous "healthy" state. The preservation of consistency prevents the system being in an undesirable state once the transaction is complete; for example, once a node has been configured, it is possible to configure it again. Isolation prevents interference between transactions; simultaneous access to data by multiple administrators may cause inconsistencies. Durability ensures that once an instruction has been committed, changes are not lost; for example, assuming nodes recover after failure, a node that fails after a transaction has completed does not revert to a pre-transactional state on recovery.

Since the community membership is dynamic, it cannot be assumed that a community member will not leave during a transaction; therefore a mechanism will be provided which will allow a transaction to complete successfully even if a community member leaves mid-transaction. Below is an example of the use of transactions in the management of a programmable network.

If a number of 'diffserv' [29] capable programmable network routers need to be configured so that some classifications of data are forwarded along a particular route, failure to update all of the nodes is likely to result in undesired routing. This may have serious repercussions if the misrouted data is mission critical. It is possible to avoid such situations if the configuration is done with transactional properties: either the entire job is done or no changes are made to the target community node's configuration.

2.6 Summary

This section has specified the fundamental mechanisms that the middleware must support for it to be used to manage a community of programmable network routers, and hence the programmable network itself. Management is the process whereby nodes are configured by administrators by sending messages over the network that interconnects them. The middleware should make it possible for a large number of programmable network routers to be managed. The network management could be autonomous or done by a human administrator who interacts with some (perhaps graphical) user interface. The middleware should support both administrator initiated configuration and target initiated configuration; this allows the programmable network to be configured in a proactive and reactive manner. The middleware is not required to understand the configuration messages it delivers, this means that consistency and model checking – if required by the programmable network – must be incorporated into the programmable networking software that uses the middleware.

For a programmable network to be effectively managed, it must be possible for all of the nodes in a community to communicate with all the other nodes. There must be some mechanism which allows community members to discover other nodes. The community is dynamic in that nodes can join and leave at any time; the location of community members and administrators is unlikely to be static. There are four delivery paradigms that must be supported: unicast, multicast, broadcast and publish-subscribe. For each of these paradigms, the middleware should allow the user to assume that once sent, a message will reach its destination. The delivery of messages must be done with a high quality of service, but real-time deadlines will not be met.

The middleware should run over Internet Protocol and be capable of configuring a heterogeneous network which spans public networks, private networks and administrative domains. It should be possible to differentiate between all of the community members, even if multiple nodes share the a common IP address.

3 Security

There are three aspects of security which strongly relate to the middleware of a programmable network: access control, authentication and privacy. In this section, each of these is discussed. Although this section does not cover the details of 'denial of service' or 'replay' attacks, the middleware should attempt to

prevent them; this should be addressed at the design phase.

3.1 Access control

Access control is the term given to a mechanism that can restrict access to functionality based on the actor requesting its use. There are two types of access control relevant to programmable networks built using the middleware proposed in this paper: community membership control and functionality control. This subsection addresses both models in turn.

Community membership control is the mechanism which allows only a certain set of nodes to become a community member. For reasons previously discussed, access control cannot be based on IP addresses; therefore digital certificates [30] may be used instead. Every community member must share a common access control policy: a node which is denied access to the community at one access point should not be granted access by another. However, it cannot be guaranteed that a node that is granted access at one point is not denied access at another; the reasons for this are outlined in the fault tolerance section. An example of how a community access control mechanism could work in a programmable network system is described below.

The owner of a network constructed using programmable network routers can be managed by a publicly available management tool which incorporates a graphical user interface. In order to prevent unauthorised nodes form accessing community members through the middleware, the community membership access control mechanism is used. The community is configured such that only nodes that possess particular digital certificate may join the community; a digital certificate must have previously been issued by some other mechanism. These access control mechanisms could be based on the functionality provided by the Public Key Infrastructure [35].

Since the middleware is a foundation onto which programmable networks are built upon, the middleware is not aware of the programmable network's functionality. This means that the middleware cannot control the access to the functionality of a community. Access to the programmable network's functionality is, if necessary, restricted by mechanisms provided by the programmable network's designer. The middleware need only provide the transport mechanisms for these access control policies. The designer of the programmable network should bear in mind that if per-node access policies are used, some security policies might be violated due to transitive access rights caused by functionality proxies. The below example shows what functionality access controls may be put in place in a programmable network. In a community consisting of programmable network routers and administrators, it is likely that only a limited number of nodes are allowed to install dropping mechanisms, whereas all administrators - providing that they do not reside on routers - may retrieve information about what dropping policies are in place.

3.2 Authentication

Authentication is a process that can be used to determine if data originated from a particular location and whether or not the data has been modified since it has been 'signed'. Authentication is particularly useful (and popular) when communications are done over non-proprietary network, where there is possibility that message source may be spoofed or the data contained in a message may be modified in transit. If it is not possible to determine where messages originate, the access control mechanisms may be circumvented. For example, a rogue administrator may try to access community members by sending messages that claim to be from a different source; this type of attack is known as spoofing. It is possible to prevent this kind of attack by using authentication mechanisms. The receiver of an authenticated message can be sure that the message has not been modified and also its source. This means that messages cannot be captured and modified in transit.

The middleware should provide an authentication mechanism that can both digitally sign and verify messages. The designer of the programmable network is responsible for providing the functionality which reacts to a message failing the verification process (i.e. it is not authentic). It should be noted that it is not a good idea to authenticate based on IP address; the two reasons for this are that IP addresses can be spoofed (as described in [31]) and because there can be duplicate IP addresses (as discussed in the 'network architecture' subsection).

3.3 Privacy

When it is necessary for the contents of a message to remain secret, encryption can be used to maintain a message's privacy. This paper assumes the term 'privacy' to mean that only the intended recipient of a message is able to read it. If messages are routed through intermediate community members, or if an eavesdropper sniffs a channel (as described in [32]), the only information that is available is the source and destination of a message, this means that it may not be possible to use transport level encryption mechanisms, such as SSL [33] and TLS [34], to guarantee privacy in the middleware. Most systems (including SSL, the industry standard encryption model) that are able to maintain privacy when transferring messages across a network assume that, to maintain privacy, it is not necessary to encrypt the source and destination of the data transfer; this project also assumes this.

Of all the security mechanisms discussed in this paper, privacy seems to be the least useful in programmable network management. However, privacy should be incorporated into the middleware as there are some

organisations who may require that the configuration of the programmable network must remain a secret.

3.4 Summary

In this section, the security mechanisms required by the middleware are discussed. It is shown that community membership, and hence the ability to manage a programmable network, is only be available to authorized parties. Since the middleware is not aware of the programmable network's functionality, it can not be controlled. The middleware provides authentication mechanisms to enable the programmable network router to determine the source of a management instruction and to ensure that the message has not been modified in transit. The middleware does not include functionality which is triggered by the occurrence of a message that cannot be authenticated; the actions must be defined by the programmable network software. In addition to access control and authentication, the middleware must provide privacy mechanisms which allow messages to remain secret when transmitted. The privacy mechanisms should prevent both packet sniffers and message sniffers (i.e. rogue community members) from compromising the privacy of transmitted data. Many, of the issues outlined in this sections can be addressed by the Public Key Infrastructure [35]; however, decisions regarding the implementation of the security mechanisms will should be addressed in the design phase of this project.

4 Fault Tolerance

Fault tolerance is the process whereby a system can continue to behave normally even when some error occurs. The laws of probability are such that: as the number of nodes in the community increases, so too does the probability that a failure will occur within the communities [36]. Since one of the foci of this project is to create a scalable middleware, fault tolerance issues must be addressed to ensure that the middleware functions correctly even when unexpected events occur, thus maintaining a high availability of the management mechanisms. The occurrence of faults will often result in event notifications being generated. Fault notification messages should indicate facts about why an operation failed but make no attempt to state why a failure occurred. The middleware is not concerned with the availability and correct functioning of the programmable network routers; rather, it is only concerned with providing fault tolerance for the management mechanisms.

In this section, the fault tolerance issues relating to security, router configuration failure, community member failures and dependency failures are discussed.

4.1 Security

If a failure occurs in a security mechanism, it may not be possible to configure the community in the normal way. There are three ways a security mechanism failure can affect the access control mechanisms of the management middleware: an administrator may be granted access it should not have; an administrator may be denied access it should have; or the fault does not affect the security of the programmable network.

It is a fair assumption that, if access control policies are not available (or invalid), administrators should be denied access to functionality rather than be allowed access. This means that the entire community will remain secure, even when serious faults occur. If access is denied due to a control mechanism fault, a notification message should be sent to the administrator who requested the operation. The notification should, rather than stating that there is a fault with the access control mechanism, state that access is denied. This is because it cannot be guaranteed that that the middleware will be aware of the fault.

If a failure occurs in the authentication mechanisms, it may not be possible to authenticate or the messages that are received. As with the access control mechanism failure: a node should not notify the sending process that the authentication has failed, not that the authentication mechanism has failed; the action taken in response to this should be defined by the designer of the programmable network.

Failure in the privacy mechanisms should not result in the privacy of a message not being maintained. If it is not possible for secure communalisations to take place between community members and the distribution requires it, the middleware should notify the software using it that the requested functionality is not available. The action taken in response to this should be defined by the designer of the programmable network.

4.2. Router Configuration failure

As previously discussed, the middleware must be capable of managing programmable networks that reside on large public networks such as the Internet where it is likely that non-proprietary network hardware (i.e. routers) separates community members. The middleware should be able tolerate both third party router failures and third party router configuration problems. This subsection outlines both of these fault tolerance mechanisms.

In order for a message to be transferred between two points, there must be a route between them. In large networks, there are usually many routes between any two points. If a fault occurs on a router or a link between routers, data must be routed in an alternate route if it is to reach its destination. The IETF [37] have outlined protocols and standards [20, 21, 38] that most 'off the shelf' level three routers support; this enables routers to discover router failure and route data accordingly. Since the middleware should be designed to work over the Internet, the functionality provided by IP routing protocols can be relied on to allow the middleware to cope with intermediate router failure.

Routers may be configured so that the management/configuration data travelling between some community members is miss-routed or dropped. From the point of the middleware, there is a fault in a router's configuration. If a configuration fault is present: data will not be routed around the faulty router because the fault is not visible to the network level routers. This means that the middleware should be capable of tolerating miss-configured routers. There are two reasons why consistency and model checking can not be used to prevent configuration faults: firstly, the configuration of third party routers cannot be controlled; secondly, the middleware is not aware of the programmable network's functionality and configuration. It has already been stated that the community may be distributed over the Internet where it is likely that many third party routers will separate community nodes, so the middleware should be capable of tolerating faults in the configuration of third party nodes that separate community members.

4.3 Community member failure

There are many ways in which a community node can fail. Community member failure is concerned with errors occurring in a node which is a part of the community; faults caused by router failure are not in this scope. It is not possible to identify all possible failures in polynomial time, let alone provide a means of tolerating them; therefore fault tolerance mechanisms should deal with a particular scope of failures rather than individual faults.

It is often very difficult to determine the location of an error that caused a fault to occur. In a community which uses the middleware proposed in this paper, the locations at which errors may be present can be grouped in to four levels: hardware (lowest level), execution environment, middleware and application (highest level). If a fault occurs at one of these levels, the levels above are also affected. For example, if an un-tolerated fault occurs at the execution environment level: the middleware and application may be affected; however the hardware level will not be affected. Errors at any of these levels should affect lower levels in that fault tolerant mechanisms do not take effect and failures do not occur. Clearly, the middleware cannot rectify problems caused by errors at the application level, but can tolerate faults at or below the middleware level. Hardware faults may result calculations generating incorrect or no results due to errors in the hardware; these errors includes chip burnouts, power outages and transient faults caused by (say) electromagnetic interference. Execution environment failures are caused by errors in the platform the middleware runs on; this includes the operating system (such as Linux [39] or Microsoft Windows [40]), the run-time environment (such as the Java runtime environment [41]) and dependency failures (discussed in the next subsection). These errors are due to poorly designed systems, incorrectly implemented software or erroneous development tools (i.e. the compiler used to build the execution environment; for example the Borland [42] or GCC complier [43]). In addition to errors at lower levels, failures at the application and the middleware level can be caused by erroneous development tools, incorrect configuration and incorrect.

The middleware may only be capable of detecting failures at, and below, the middleware level. This project is not concerned with tolerating faults within community members; rather, it is more concerned with enabling the community to cope with the complete failure of one or more community members. For this reason, a community member will be considered to have failed if the middleware (either directly or indirectly) detects any kind of error at a node's middleware, execution environment or hardware level. The result of this is that faults will result in a community member becoming unavailable.

This project assumes that if a node fails, it will eventually recover. To prevent malicious or malfunctioning nodes from ejecting other community members, a node must leave the community to explicitly requesting membership termination. Since few computing systems are able to complete the recovery procedure without interaction with a human or external system, the middleware should be capable of notifying the appropriate administrator that a fault has occurred. However, the middleware will not require that the user of the middleware provides a mechanism to receive or act on the failure notifications. Node failure is discovered by other community members by either failing to send it a message or by realising that it is not responding to heartbeat messages (that is if it is decided during design phase of this project that heartbeat and ping messages exist in the middleware).

In the section that outlines the fundamental mechanisms the middleware must possess it is stated that a user can assume a message is delivered once it is sent. If the destination node has failed, the message must remain in the community until it the node recovers; however, if the node does not recover in a reasonable time, the sending node should be notified that the message could not be delivered.

4.4 Dependency failures

There are very few programmable networks that do not depend on functionality provided by external services. For example, programmable networks based on the CORBA [12] middleware depends on the availability of Object Resource Brokers (ORBs); programmable networks based on JMS [8] are dependent on the JMS server. If a systems dependencies are not available, is unlikely that the programmable network will operate correctly (if at all). The middleware must not have a single point of failure, and so will not be based on unreliable dependencies.

Thanks to the highly active fault tolerance research community, there is a high level of fault tolerance in most dependencies. However, it seems that many of these sophisticated fault tolerant systems depend on the availability and correctness of a very small number of Domain Name System [44] (DNS) servers. The middleware should, if possible, mot rely on dependencies such as these.

4.5 Summary

This section has presented the fault tolerance mechanisms that are required in the middleware. Fault tolerance is the process whereby the occurrence of errors in the community does not result in compromised security. The middleware is not concerned with the availability and correct functioning of the programmable network routers; rather, it is only concerned with providing fault tolerance for the management mechanisms. It is important to realise that consistency and model checking can not be used by the middleware to prevent errors occurring, therefore the faults must tolerated.

The middleware should guarantee that, even when faults occur, only authorised nodes are community members. The failure of the authentication or privacy mechanisms should not result in security breaches. In addition to the middleware mechanism failures, errors that exist in third party systems, particularly routers and third party services (dependencies), could adversely affect functionality of the management middleware. The middleware fault tolerance mechanisms should try to ensure this does not happen. If it is not possible to tolerate a fault, the middleware should be able to notify some external system (for example a human operator) of the problems.

5 Usability

It is unfortunate that many of the popular middleware designs seem to have overlooked usability issues; this project does not intend to follow suit. There are three main areas where usability is important: installation of the middleware, configuration of the middleware and also how easy it is to use the middleware.

At the time of installation, the systems on which a middleware is dependent must also be installed. Most software distributions seem to include dependencies in the package; however, some do not (most open source software requires the system administrator to acquire, install and configure dependencies before the install can be done). The middleware being proposed in this document should, if possible, provide all of the dependencies above the network stack.

The complexity of many middleware designs sometimes results in a middleware which is difficult to configure. It is a goal of this project to produce a middleware that is easy to configure. If possible, the configuration of the middleware should be totally automated.

The middleware's Application Program Interface (API) should be clear and straightforward to use. The user of the middleware should not be required to learn how of the middleware works, rather, only the way in which the functionality is used needs to be understood. Fault tolerance mechanisms should be completely transparent to the programmer; transactions and security mechanisms should not.

6 Related work

There are many programmable network projects that have roots in either Active Networks [52] or OpenSig [48] projects. Currently, most of the work in the programmable router field seems to focus on application level active networking [53]. A good overview of all areas of programmable network paradigms is given in [54], and many of the well known research projects in this area are compared in [55]. More recently, a number of new projects, including Promile [45], Click [46] and Android [47], have focused on discrete active service programmable networks.

To the knowledge of the author, this is a the only research project concerned with the management of a large heterogeneous programmable network. However, research groups at Sussex university [56], Lancaster University [57], Sydney University [58], Imperial College [59], University College London [60] and BTExact [1] seem to be moving in a similar direction. In addition to work done in the field of programmable networks, this project may also benefit from work done in the area of management of overlay networks such as MBone [61] and ABone [62].

There several relevant conferences that publish work relating to this project; these include OpenArch [49], IWAN [50], Middleware [51] and OpenSig [48].

7 Conclusion and Future Work

This paper has stated the requirements to which the middleware must conform together with the rationale behind each requirement. There are many programmable network research projects, but the vast majority focus on the software architecture rather on management. The projects that are concerned with management tend to incorporate the management into the programmable network router and are content with being capable of only managing a single node. To the authors knowledge, this is the only project that is concerned with the development of a general purpose middleware for the configuration of programmable networks consisting of a large number of heterogeneous programmable network routers.

Now that the requirements of the middleware have been considered and formally documented, our attention can be focused on the design of the middleware. As indicated in this paper, there are a number of issues that must be resolved regarding the

architecture of the middleware. One of the key goals of the design phase is determining whether or not any existing middleware can be used to construct the management middleware proposed in this paper.

9 References

[1] BTExact Technologies, http://www.labs.bt.com/

[2] XML, http://www.w3.org/XML/

[3] Schema, http://www.w3.org/XML/Schema

[4] Goldszmidt, G., Yemini, Y. "Distributed management by delegation", Proceedings of the 15th International Conference on Distributed Computing Systems, 1995 pp. 333–340.

[5] Yemini, Y., Konstantinou, A.V., Florissi, D. " NESTOR: an architecture for network self-management and organization", IEEE Journal on Selected Areas in Communications, 18(5): 758 –766, May 2000.

[6] Xlinkit, http://www.xlinkit.com/

[7] Holsman, G. "The Spin Model Checker", IEEE Trans. on Software Engineering 23(5):279-295, 1997

[8] JMS, http://java.sun.com/products/jms/

[9] DyNet, http://www.socketsystems.co.uk/dynet/

[10] White, J. "Telescript Technology: The foundation for the electronic market place", General Magic white paper, 1994.

[11] Aglets, http://www.trl.ibm.com/aglets/

[12] CORBA, www.omg.org

[13] RMI, http://java.sun.com/products/jdk/rmi/

[14] Cheriton, D., Deering, S. "Host Groups: A multicast extension for datagram Internetworks", Ninth Data Communications Symposium, ACM Computer Communication Review, Vol. 15, Sept. 1985.

[15] Postel, J., Reynolds, J., "File Transfer Protocol (FTP)", RFC 959, October 1985.

[16] JNDI, http://java.sun.com/products/jndi/

[17] JINI, http://java.sun.com/products/jini/

[18] Wahl, M., Howes, T. and S. Kille, "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997

[19] Postel, J. "Internet Protocol", RFC 791, Sept 1981.

[20] Moy J., "OSPF version 2", RFC 2328, April 1988.

[21] "A Border Gateway Protocol 4 (BGP-4)", RFC 1771, March 1995.

[22] Leech, M. et al. "SOCKS Protocol version 5", RFC 1928, March 1996.

[23] Egevang, K., Francis, P. The IP Network Address Translator (NAT)", RFC 1631, May 1994.

[24] The Internet Corporation for Assigned Names and Numbers, http://www.icann.org/

[25] NTL, http://www.ntl.com/

[26] Telewest, http://www.telewest.co.uk/

[27] British Telecom, http://www.bt.com/

[28] Dirckze, R.A., Gruenwald, L. "Nomadic transaction management", IEEE Potentials 17(2): 31-33, April-May 1998.

[29] Blake, D. et al. "An Architecture for Differentiated Services", RFC 2475, Dec 1998.

[30] Diffie, W. "The first ten years of public-key cryptography", Proceedings of the IEEE 76(5): 560–577, May 1988.

[31] Daemon9, Route, Infinity "IP-spoofing Demystified", Phrack Magazine 7(48), June 1996.

[32] McDonel, J. "Defeating Sniffers and Intrusion Detection Systems", Phrack Magazine 8(54), Dec 1998.

[33] SSL, http://www.netscape.com/eng/ssl3/

[34] Dierks, T., Allen, C "The TLS Protocol Version 1.0", RFC 2246, Jan 1999.

[35] Aresenault, A., Turner, S. "Internet X.509 Public Key Infrastructure: Roadmap", PKIX Working Group Internet draft, Jan 2002.

[36] I. Mitrani, "Probabilistic Modeling", Cambridge University Press, 1998.

[37] The Internet Engineering Task Force, http://www.ietf.org

[38] Hedrick, C. "Routing Information Protocol", RFC 1058, June 1988.

[39] Linux, http://www.linux.org/

[40] Windows, http://www.microsoft.com

[41] Gosling, A., Joy, B., Steele, G. "The Java Language Specification", Tech. Rep., Addison Wesley, 1996.

[42] Borland, www.borland.com

[43] GCC, http://gcc.gnu.org/

[44] Mockapetris, P. "Domain Names - Concepts and Facilities", RFC 1034, Nov 1987.

[45] Rio, M. et al. "Promile: A Management Architecture for Programmable Modular Routers", In the proceedings of OpenSIG 2001, September 2001.

[46] Kohler, E. "The Click Modular Router", ACM Transactions on Computer Systems 18(3): 263-297, August 2000.

[47] Android, http://www.cs.ucl.ac.uk/research/android/

[48] Open Signalling Working Group (OpenSig), http://www.comet.columbia.edu/opensig/

[49] Open Architectures and Network Programming (OpenArch), http://comet.columbia.edu/activities/openarch/

[50] International Working Conference on Active Networks (IWAN), http://dblp.uni-trier.de/db/conf/iwan/

[51] IFIP/ACM Conference on Middleware, http://www.ifip.org, http://www.acm.org/

[52] DARPA Active Network Program, http://www.darpa.mil/ito/research/anets/

[53] Fry, M. and Ghosh, A. "Application layer active networking" HIPPARCH '98 Workshop

[54] Fisher, M. "D5: Android Active Networking Architecture", Android Consortium, June 2001.

[55] Campbell, T. et al. "A Survey of Programmable networks", ACM SIGCOMM Computer Communications Review 29(2):7-23, April 1999.

[56] Software Systems Research Group, Sussex University, http://www.cogs.susx.ac.uk/lab/softsys/

[57] Distributed Multimedia Research Group, Lancaster University, http://www.comp.lancs.ac.uk/computing/research/mpg/

[58] Distributed Computing Research Group, University of Technology, Sydney, http://it.uts.edu.au/
[59] Distributed Software Engineering Group, Imperial College, London, http://www-dse.doc.ic.ac.uk/
[60] Software Systems Engineering Group, University College London,
[61] Eriksson, H. "The Multicast Backbone", Communications of the ACM 37(8):54-60, Aug. 1994.
[62] Berson, S., Braden, B., Riciulli, L. "Introduction to the ABone", Information Sciences Institute, June 2000, http://www.isi.edu/abone/
[63] Emmerich, W. "Engineering Distributed Objects", Wiley, 2000.

11 Acknowledgements

Firstly I would like to thank my supervisor Wolfgang Emmerich and the Promile research group based at University Collate London for their continuing support of my research and for the stimulating discussions relating to the area of programmable networking. Paul McKee and Mike Fisher from BTExact have also been very supportive, especially with regard to the place of programmable networking in industry and existing work in this field. Thanks also go to EPSRC and BTExact for funding my research.

Andrew Hughes is currently perusing a PhD in the Department of Computing Science at University College London, his research is focused in the area of programmable networks. He is one of the first students to be based at the new UCL campus at Adastral park which is designed to strengthen the collaboration between the university and BTExact Technologies. Andrew graduated from the University of Newcastle-upon-Tyne in 2001 with an honours degree in computing science. In 2000, he co-founded a software development company that works in the area of virtual networking.