Management of Active Networks

Marcus BRUNNER, Bernhard PLATTNER

Computer Engineering and Networks Laboratory, TIK
Swiss Federal Institute of Technology Zürich (ETH)
Gloriastr. 35, CH-8092 Zürich, Switzerland
E-Mail: brunner@tik.ee.ethz.ch


Abstract

Active networks provide the possibility to perform computations within the network. This feature introduces greater flexibility, but more complexity as well. Therefore, a sound management of active networks is a prerequisite to the acceptance of such networks. Management of large complex systems can be improved with the introduction of further abstraction levels and with moving the responsibility for the management down the line. Network management applications have two basic tasks to perform: operation of a node and control of message execution. In active networks there is a need for debugging functions a user has injected into the network. For operation and debugging, the monitoring of events occurring within an active network, and visualizing it, helps performing these tasks. Utilized and well scaling mechanisms for monitoring active networks are node clustering, event abstraction and filtering which have been applied to the presented prototypical implementation. We further propose a framework for monitoring and managing the active network.

1 Introduction

End systems of a network are open systems in the sense that they can be programmed with appropriate languages and tools. In contrast, intermediate network nodes are very often closed, integrated systems, whose functions are determined through long lasting standardization processes. Traditional data networks transport bytes from one end system to another. The intermediate system neither modifies these byte or bit streams nor are they sensitive to their contents. Because the processing in intermediate nodes currently is extremely limited, it consists of header processing in packet-switched networks and signaling in connection-oriented networks.

Active networks break with this tradition by letting the network perform customized computation on the data. This implies that an active network node has two tasks to perform: (1) computation on user data and (2) individuals can inject functions into network nodes, thereby tailoring nodes’ processing to be application-specific. Deploying software in these intermediate nodes should be as simple as launching an end system application.

With the greater flexibility gained through the possibility to program the network, a drawback on the ease of manageability of the network has been introduced. However, a huge potential to overcome several problems of traditional network management systems are achieved since, such as scalability in network size, static functionality, reliability of the management application, and the problem of micro-management.

With the introduction of active networks, network architecture gets very flexible, but with this flexibility an increase in complexity comes along with. Therefore, we identified an even stronger need for network management which is not handled sufficient in today’s networks. An efficient and scalable resource and network management is mandatory, if active networks are becoming a competition for today’s networks. The key factors in the management field is the problem of scalability in large networks management and the static functionality. To manage large systems two basic approaches are considered: (1) abstraction of low level structures and (2) delegating the responsibility to a lower level. The static functions are replaced with dynamic ones, with the possibility to inject programs into the network. As everyone can inject programs and users programming such functions want to debug them and want to observe their performance.

For debugging and management tasks visualization tools help understanding the execution of a program and the overall network performance. It is widely accepted that visualizing data is more useful as larger amounts of data have to be presented to a user. Therefore, visualization will help to understand active networks with a huge amount of messages transmitted. This understanding is mandatory to improve the network architecture, injected active messages and the management of an active network.

This paper is organized as follows. Section 2, describes an active network architecture for studying resource and network management in active networks. In section 3 mechanisms for operation management and debugging of active messages are presented. Issues on the monitoring of active networks are depicted in section 4. A framework for monitoring and managing is outlined in section 5, which is followed by the conclusion and a brief discussion on further work.

2 Architecture of an Active Network

We propose an architecture of an active network which is used as a prototype for exploring issues of resource management and network management. This architecture does not focus on the performance or security, but it provides an environment to test different ideas. Since at the beginning of the work no architecture has been available for active networks, a set of general mechanisms, described later, has been developed which could
be refined and may be migrated to others.

2.1 Network Model

A network consists of nodes which are connected via links. Moving entities in the network are “active messages”, also called “capsules” in [1]. They are comparable with packets in traditional data networks, despite the fact that they are not only data, but also programs. An intermediate node is processing an incoming active message accessing the environment of this node. Afterwards, the message chooses an outgoing link to leave this node.

A special type of nodes are end system nodes. On the node at the border of the network an active message may not only choose to leave the network via a link, but it may also choose to be delivered to an application or to access a displaying device.

2.2 Protocol Architecture

Despite of the fact of manifold definitions for a protocol, refer to [2], in our case a protocol consists of a set of active messages which may be of similar or different type. Therefore, a protocol is the base unit for reservation, classification, and protection. This could be done also on a per active message basis, but if more than one type of active messages is involved in a protocol, it is impossible that one of these messages can profit from resources which another one does not consume. If resources have to be allocated to only one message type, more than one protocol for a communication can be specified easily.

2.3 Node Architecture

In the network model depicted above, nodes receive packets and process them with the information stored in the node, such as state. This results in one or more packets being transmitted on an outgoing link. This functionality requires four major components within a node as depicted in Figure 1, (1) the control of incoming messages, (2) the control of the outgoing messages, (3) message processing, and (4) memory to store information.

The difference compared to traditional networks is based on the processing component, where nodes have static pre-installed functions processing incoming packets. However, active networks allow users to install their personalized functions in any network node. The proposed approach proceeds one step further. The function to be processed on the network node

\[\text{Figure 1: A Generic Network Node Architecture}\]

2.4 Interface of a Node

The active message running on a node need to access the node over an interface. The tasks a message can execute are many fold. Bet the interface can be grouped into different functional areas: 1) execution, 2) information, 3) configuration, and 4) organization.

\[\text{Figure 2: Interface of an active network node}\]

3 Network Management

The paradigm of management by delegation [4] can be used to move management functions to the data rather than moving data to these functions. Delegation fits very well into the concept of active networks. Active

---

\(\text{‡} \) Traditionally, a protocol defines a set of rules for communication. In our case these rules are specified by a programming language.

\(\text{‡‡} \) In the integrated services network community the entity for resource reservation is termed a “flow”.

---
messages can retrieve data about the state of a node process this data and take appropriate action.

There are three important issues associated with the management of active networks. First, the management of the network node with its resources. Second, the management of configurations, and faults. Third, the management of active messages which essentially results in the monitoring and debugging active messages flowing through the network.

3.1 Node Management

Here the focus is drawn to the operation, configuration, and fault management of a node. These tasks comprise functions which are not visible to normal active messages flowing through a network node. They are strictly utilized by the node’s system manager to ensure an optimal operation of the node. Configuration possibility are the installation of special node components or setting run-time parameter of the components. E.g. you may change the out-bound link scheduler to have another scheduling policy on this link or you may change the minimum time quantum of the scheduler.

In many cases such operation changes have to be done, not only on one node, but on several nodes in the network.

3.2 Resource Management

If an active network shall support guaranteed services, resources of a node have to be managed in a fashion that guarantees can be fulfilled by the network or that the messages can provide a guaranteed service to a client. Since existing resources are used by all active messages on a node, it is fundamental and mandatory to isolate the resource consumption of each group of messages (protocol) from other groups. If this is not the case, there is no way to protect a group of messages from being disturbed by other messages overconsumming these resource.

A resource not consumed should be shared fairly among protocols, as defined in subsection 2.2, competing for that particular resource. Anyway, for different classes of network traffic, where a class represents some aggregate of traffic streams being grouped according to their administrative domain, protocol, traffic type or other criteria, an unused resource shall be provided to a class member using this resource, but not to members of different classes. Therefore, the support for hierarchical link sharing has already been proposed by [5]. In this case each resource has an associated class hierarchy, specifying the allocation policy of this resource. This hierarchy allows for building virtual private networks. According to the traffic types of the messages also the resource scheduling has to be different, which leads to the need of changing the scheduler.

3.3 Debugging of Active Messages

Empirical evidence shows that programmers spend the majority of their time on program maintenance tasks such as debugging [3]. A first step in all debugging purposes is to gain the understanding of the program at hand. Since active messages are programs flowing through the network, it is an even more difficult task to debug them. A more frequently applied support is a graphical visualization to present, follow, and analyze the execution of a distributed computation. The same holds for active messages. An event-based approach is typically employed for debugging distributed programs. In the following, events and their collection is discussed.

Observation of Events

An event constitutes the lowest level of observable behavior. It occurs each time an to be observed action is executed. Each event represents a specific action and is typed according to the action this event represents. Thus, different events of the same type may occur, but they are separated in time, by messages, or by message type. An event is considered to have no duration, which, of course, is an ideal assumption.

Actions of interest can be classified in two groups: (1) Those arising from the execution of a active message on one node, for example scheduling decisions, arrival of a message, or transmission of a message, or (2) those arising from actions happening within the node without or on unknown relation to a message, for example errors, overloads, and system errors. Events in the first class originate at the interface of the run-time environment of a node which the message is accessing during its life-time on this node. Second class event originate in active components of a node.

Time

With the occurrence of an event time is associated. This time is interesting, because (1) ordering events and (2) studying temporal aspects of the system are feasible. In distributed systems two different principles to deal with time exist. Events may be ordered partially with the introduction of the ‘happens before’ relation proposed by Lamport [6] and [7]. This means, if an event A ‘happens before’ event B, event B may be dependent on event A. Another approach implements a global time base as accurate as possible [12]. Typically, timestamping techniques consist of two parts: (1) a timestamping algorithm (virtual or real clock) and (2) a timestamp test. The algorithm describes how to assign timestamps to an event and the test determines which event happens before another one.

Collection of Events

Since events occur in a node, the interface of this node needs a function to register an object with a predefined interface to be called on the occurrence of this event.
Furthermore, it has to be specified which events a message is interested in. Another entity has to deal with timestamping, if necessary for further calculation on events. According to the precision of the clock and the placement of the timestamping mechanism for events, the accuracy of the timestamp is determined. Timestamping the event as near as possible with the real occurrence of the action it represents, achieves best results. Therefore, the optimal solution for an event to receive accurate timestamps is to install an opaque clock with an abstract interface into the node, delivering the current time. These allow for introducing different timestamping algorithms.

Identifying Messages
If events of the same message on different nodes have to be correlated, which is normally the case in the debugging scenario, it has to be possible to identify the message. If the nodes itself are globally unique identifiable, the messages can be identified with identity of creation node and a number which is unique on this node. This information has been transported along with the message. Another approach is to identify the message only between the sending and the receiving node. This requires the possibility to detect, if the message is lost or corrupted on the link.

4 Monitoring of Active Networks
For operation, performance measurements, and debugging of active messages, visualization tools and techniques are promising. They allow for a better understanding of gathered network data. The understanding of the network is very important, since the active network technology is new. For the operation task the interesting views are the network topology including the information, how networks actually perform and indicating nodes with a failure situation. However, for the debugging task additional low level views are interesting. Low level in the sense that a programmer focuses his attention to a small set of messages, but exactly determines which executions these messages perform on their lifetime.

The visualization of data greatly improves the understanding of a topic. The more data are to be presented to a user, the more a visualization tool supports the comprehension of the state of affairs. If the new technology of active networks is evolving, it is very important to employ and improve it. Still, the problem of an ever increasing network size remains in addition to a concomitant increase of collectable network measurement data. This large amount of data for visualizing the system calls for a reduction of the data to an amount which is feasible to be displayed on powerful graphic workstations. Concerning the debugging case, the programmer debugging his code, first wants to have a global overview of the execution of the program [9]. This can be achieved by viewing the execution at a very high abstraction level. Further detailed understanding is more difficult, but supported by an isolation of interesting parts of the execution to be examined at a lower layer of abstraction. On different layers in debugging, a different kind and variable number of events are required.

To achieve a high-level view three mechanisms are available: (1) grouping low-level events to form “abstract events”, (2) grouping single nodes to a cluster of nodes, and (3) filtering out events which are not required in a high level view. All these mechanisms can be supported well with the introduction of an active network architecture, because they distribute the computation of the visualization tool from the displaying workstation into the network.

4.1 Event Abstraction
The motivation of event abstraction is to reduce complexity by omitting unwanted details. The abstraction should not introduce spurious precedences or hide real ones. The most general potential definition of an abstract event is an arbitrary subset of events in a session. As mentioned before, of particular importance for the analysis of the distributed processing of active messages is the relation “happens before” between events [6]. This relation induces a partial order of the event set. But the relation holds for atomic events only, abstract events have a duration and probably happen on different nodes. Therefore, a number of schemes for event abstraction have been proposed, early ones in a more ad-hoc manner without firm theoretical foundation. In [8], the notion of molecules and atoms defining sets of primitive events with certain properties is introduced and a precedence relation on such abstract events is defined. [10] presents an abstraction algorithm on a causality graph with events as graph nodes to collapse nodes, e.g., events, into supernodes, e.g., abstract event. [11] proposes convex abstract events as good automated abstraction of events to enhance the expressiveness of the display of such abstract events. These approaches have in common that a possibility for the automated calculation is available. Low level events have to be collected from more than one node to form an abstract event.

For example, assume a remote procedure call (RPC) first performing a lookup on a server to determine, where the function is implemented, afterwards calling the procedure and waiting for the result (cf. Figure 3). The RPC considered as a whole is an abstract event in the execution of a program, the host lookup and the call themselves are abstract. Low level events are the arriving and leaving of messages on intermediate nodes. To construct this abstract event, the collection of the arrival and departure of messages of the RPC-protocol has to be performed. In addition involved
nodes can combine these information and create an abstract request event. E.g., if the RPC-protocol is not working correctly, the top-level view shows the RPC event by starting a debugging session. Proceeding on a lower layer, for example, the call event never occurred, determining that the host lookup has failed. Stepping deeper in this hierarchy more details and, finally, the error can be detected.

4.2 Node Clustering

Since in modern networking environments operators deal with many nodes and links involved in the execution and transmission of active messages, it is not possible to visualize all these nodes together. Therefore, a clustering of nodes supports a structured overview. For monitoring purposes a clustering of highly interconnected nodes to one collapsed node is suitable, whereas for debugging, for example, a point-to-point protocol the programmer may choose to collapse all intermediate nodes in one visible node, only showing two end-system nodes with one intermediate in-between. If the concept of subnetworks is introduced in the active network environment too, a clustering of subnetworks can be performed easily. In general, finding highly interconnected nodes is a very complex task. The problem to be solved is to find an appropriate and automatically computable clustering. However, this is the problem of a programmer implementing a visualization tool. The important feature of computational resources in the network is the fact that this clustering can take place within the network and only data of the collapsed node is sent to the displaying node. This mechanism reduces heavily the load on the network and the visualization workstation.

4.3 Event Filtering

Filtering of events can take place at different locations on the path from the source to the sink. First, a selection of all possible events on a node is collected. This loss of certain information happens from the beginning, but reducing a data stream at the source is the best way to save resources in the following computation and transmission operations. Second, during the collection of events from different nodes, a filtering can take place. For example, due to an overload situation in the node or at the displaying end-system or if more than one display system is used for displaying a more precise view and an overview, filtering takes place at the branching node of the multicast tree. Third, on the workstation displaying events, a filtering can take place according to the preferences of the system manager maintaining the network. Naturally, this is not an ideal situation, because the information being transferred to the displaying station should not be discarded. In this case, after some time filtering should happen within the network. Therefore, the filtering has to be flexible to support different requirements of a system operator.

5 A Framework for Monitoring and Management

The main design goal of the framework is to decouple hierarchical clustering of nodes, the monitoring process, and the introduction of new services for any kind of management services into the network.

5.1 Hierarchical Clustering

To attack the scalability to the number of nodes in the network a hierarchical grouping of the nodes is suitable. So each node is a leaf in tree of clusters. Which nodes are aggregated together to a cluster, has to be open. E.g. if you want to measure performance of subnetworks, the clustering algorithm aggregates the nodes of a subnetworks. If the resources of a new connection are reserved and monitored, the cluster consists of all nodes over which this connection is running.

So for each type of monitoring or management task a different tree of clusters is constructed, and held over some time. Each cluster has with it a level number identifying his depth in the tree.

5.2 Dynamic Data Selection

The point to get and select node information is on the node itself. This can be done with the installation of a extraction service, which can be as dynamic as it has to be for a specific monitoring task. Typically the same service is installed on different nodes. The information
can be stored in the nodes memory and accessed later by a information collector.

5.3 Level-specific Information

For each cluster or node the relevant information can be stored in the memory of a node, representing the cluster. Access to this information can be level-specific. E.g. the question is, how many messages of a specific type are moving through which networks? The lowest level cluster gathers the number of messages of his nodes and stores it in his memory. The next upper node gathers the summed up values of his lower cluster. Now the manager may choose on which level he wants to get the information. This is dependent on the number of clusters, the bandwidth of the last link to the manager, the display or store capacity the managers station has.

5.4 Message types

In this framework there are different predefined messages to design and implement monitoring or management applications.

- Installation of a service in a specific level in the cluster hierarchy. The service can be a monitoring service or a management service.
- Child information collectors. They can be used to collect some information from the child clusters. This messages uses the information of the cluster tree.
- Monitor information collectors. They get the information of a specific level for a monitor of the network. This messages reads information a service or a child information collector has stored.

6 Conclusion

With the introduction of active networks, we identified an even stronger need for network management which is not handled sufficiently in today’s networks, because of the increased complexity and flexibility. Therefore, an efficient and scalable network management is mandatory, if active networks are becoming a competition for today’s networks. We presented mechanisms to manage resources and global networks. The key factor in the management field is the problem of scalability to large networks and dynamic management functionality. To achieve scalable solutions in active networks, network management is decentralized by bringing the computation nearer to the managed objects. This results in a very flexible way to carry out management tasks with the introduction of code into the network, and provides the possibility to reduce the load on the management workstation by applying mechanisms as event abstraction, filtering, and node clustering. New management functionality may be implemented without changing the nodes’ interface.

7 References