ACTIVE PIPES: SERVICE COMPOSITION FOR PROGRAMMABLE NETWORKS

Ralph Keller[1], Jeyashankher Ramamirtham[2], Tilman Wolf[2], Bernhard Plattner[1]
[1] [keller | plattner]@tik.ee.ethz.ch [2] [jai | wolf]@arl.wustl.edu
[1] Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology
[2] Department of Computer Science, Washington University in St. Louis

ABSTRACT

Active networks allow customized processing of data traffic within the network which can be used by applications to improve the quality of their sessions. To simplify the development of active applications in a heterogeneous environment, we propose "active pipes" as a programming abstraction to specify transmission and processing requirements. We describe how an active pipe can be mapped onto network resources by a shortest path algorithm, and how optimal processing sites and a route through the network can be determined. Additionally, we propose a scalable network software architecture implementing the functionality required for active pipes.[1]

I. INTRODUCTION

Active networks provide processing capabilities on routers that allow customized handling of data traffic within the network [2], [10], [13]. This allows applications to be distributed, parts of the application executing on end systems, and portions of the application running on intermediate network nodes that process the data stream. Processing in the network can be used to deploy new services like congestion control, transcoding, monitoring, and thus improve end-to-end session quality. Such upcoming multiservice information networks require sophisticated network control software capable of allocating processing resources and bandwidth to applications efficiently. This requires routing protocols capable of distributing information about the availability and usage of system resources and session configuration mechanisms that can quickly map session requirements onto available resources. One objective is to make the use of advanced services as simple as possible for end users.

However, most active networking environments require the application to explicitly specify the location by a network address where code modules need to be deployed. Thus, an understanding of the underlying network infrastructure and the system architecture of the active network is necessary. This burdens the end user and makes large-scale deployment of active network services impractical. Deploying code should be a simple task, hiding the internal details of the network from the application whenever possible. Also, optimal resource allocation should be delegated to the network, freeing the application from this task. Thus, it is necessary to have a general scheme of specifying application requirements that is expressive enough to describe typical application scenarios while simple enough to be used effectively.

In this paper, we propose *active pipes* as a programming

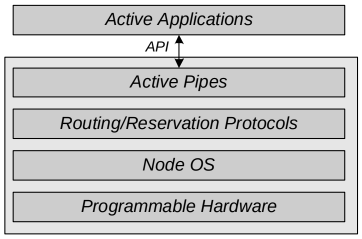

Figure 1: Abstraction layers

abstraction that allows specification of processing requirements by applications. As illustrated in Figure 1, an active pipe provides an interface between the user and the active network and uses existing reservation mechanisms for session setup. It significantly simplifies the use of active networks regardless of the node implementation, providing a crucial component missing in current active network frameworks.

The idea of specifying transmission and processing requirements is to model a connection as a sequence of functions that have to be performed on the data stream. This concept is analogous to pipes in UNIX where data can be sent through a sequence of programs. In the active networking context, each function corresponds to a code module that has to be installed on a router along the path of the connection. Additionally, the application can define constraints where such processing should take place.

This paper addresses the following three issues:

- We introduce a method for specifying transmission and processing requirements for connections over active networks.
- We demonstrate how the connection requirements can be mapped onto network resources while minimizing network costs.
- We propose a network software architecture that can implement the services required for resource mapping, routing, and connection setup.

In Section II, we describe our programming paradigm for active networks and explain how active pipes are defined. Section III illustrates how we can map an active pipe onto network resources and determine an optimal route and the location of processing sites. In Section IV we address scalability issues and demonstrate, how our scheme can be distributed. Section V presents a node architecture to implement the required functionality. In Section VI, we discuss related work and Section VII concludes this paper.

II. ACTIVE PIPE PARADIGM

An active pipe is used to describe transmission and processing requirements as a sequence of functions that have to be performed on the data stream. Each function corresponds to a code module that needs to be instantiated on a node along the path. Since appli-

[1.] This research was supported by DARPA under contract number N66001-98-C-8510.

cations can have stringent requirements on the location of processing modules, the specified processing steps can have constraints associated with them. For example an application requires sufficient processing capabilities from an active node to guarantee proper execution of the code module. In addition, the location of processing modules can be restricted to meet other specific requirements such as installation of the processing module within a given address range.

Figure 2 depicts an active pipe for a scenario where a connection for sensitive data transmission should be established between two end systems, located in different domains. The end system domains are assumed to be secure but since traffic transits untrusted nodes, encryption and decryption steps are needed in the source and destination domains, respectively.

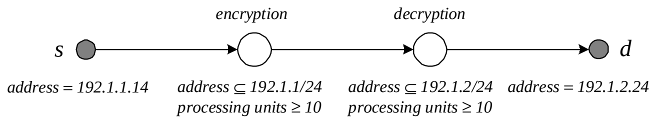

Figure 2: Secure data transmission pipe

The active pipe includes the end systems and processing functions that should be deployed in the network. Constraints are defined in form of attributes that must satisfy specific values given corresponding relations. In the network each node has a set of attributes that describe the node's static and dynamic properties. Static attributes include location attributes (network address, domain, border router etc.). Dynamic attributes describe the current work load (available processing cycles) and link load. To be considered for deployment, a node needs to meet all of the given constraints. In the example above, the encryption function must be on a node within the address range 192.1.1/24 and sufficient processing cycles need to be available. Thus, for each processing step, constraints define a subset of nodes that are qualified to execute a given function.

The active pipe abstraction defines a logical end-to-end path which includes processing sites that should be instantiated on nodes satisfying the given constraints. Next, we propose an algorithm that maps such a logical path onto the underlying physical network.

III. Mapping Active Pipe Onto Network Resources

This section describes how we can map a high-level pipe abstraction onto physical network resources. The result of this scheme is the selection of suitable processing sites as well as a path traversing processing sites in a given order. As specified by the active pipe, each processing function can have various constraints that must be satisfied by the algorithm. Also, the installation of a processing module has an associated cost, which can vary between processing sites, thus processing costs must be taken into account as well.

We start with the simplest scenario where we assume that the mapping algorithm has a complete view of the network, as is common for link state protocols such as OSPF [9]. First, we demonstrate how we can solve the problem for a single processing site and then extend our scheme to multiple processing steps that are executed in a sequence. In Section IV we discuss how our scheme can be extended to large networks.

A. Single Processing Site Mapping

In the simplest case as illustrated in Figure 3, one intermediate computation needs to be performed on a node within the specified address range that is capable of performing the required processing.

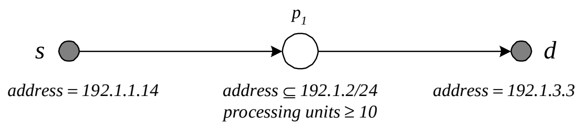

Figure 3: Active pipe with single processing step

Figure 4 depicts a sample network with various processing sites. Each processing site has an associated cost (shown as the number in the node) that needs to be taken into account when processing on that site should occur.

To map the active pipe onto the network, we must select one processing site and a path connecting the end systems through the processing site, preferably in a way that minimizes both link and processing costs.

First each potential processing site is compared with the processing step constraints in the active pipe. If a node cannot be a candidate for a given function (because it is outside the address range or not enough cycles are available), it is excluded from the set (e.g., the processing cost is set to infinity). For simplicity, we assume that all sites qualify in this example.

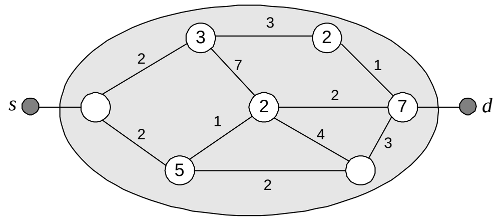

Figure 4: Network with processing sites

Formally, the problem for the single processing site case can be stated as follows. The network is represented by a directed graph, $G = (V, E)$, in which vertices correspond to routers, while edges correspond to links. Each link $e \in E$ has an associated transmission cost $c(e)$, and each node $v \in V$ a processing cost $c(v)$. The source is defined by s and the destination by d. Finally, let $R \subseteq V$ be the subset of nodes that represents sites where intermediate processing may occur, that is nodes that satisfy the constraints for the processing step. Our goal is to find a path from s to d that includes at least one node $r \in R$ while minimizing *both* link and processing costs.

Shortest path problems with more than one cost metric are known to be intractable since their computational complexity is NP-complete [11]. To simplify the problem, we assume that processing costs are scaled to match the link cost units. This is a convenience that allows us to transform the multiple metrics routing problem into a shortest path problem with just link costs as described in [6]. With the single metric assumption, the cost of the path can be expressed by the sum of all link costs plus the processing cost at the chosen processing site.

We can solve the problem with one processing site by trans-

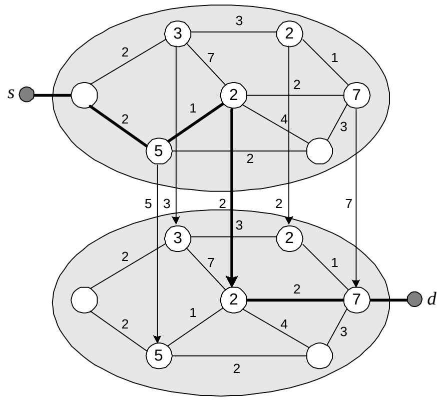

Figure 5: Layering model for single processing site

forming it to a shortest path problem on a *different graph*. We modify the graph G by making two copies which we identify as layer 1 and layer 2 as illustrated in Figure 5. For each vertex v in the initial graph, let v_1 denote the vertex in layer 1 of the target graph while v_2 denotes the vertex copy in layer 2. To model the processing function, we add edges *between* the two layers. For every node $r \in R$, where processing may occur, we add edges (r_1, r_2) in the target graph and let the link cost of (r_1, r_2) be the processing cost on node r, $c(r)$. The source node s_1 in layer 1 is the source for this new graph and the destination node d_2 in layer 2 is the destination node for the new graph. To solve the routing problem with one mandatory processing site, we find a least-cost path in the target graph using a shortest path algorithm. Once we have found a solution in the target graph, the path can be mapped back to the original graph by projecting the two layers onto a single layer. The processing is optimally performed where the path *crosses the two layers*.

B. Multiple Processing Sites Mapping

The layering model can be extended to include several computational steps. For each processing step, there are potentially multiple locations where processing may occur. For example a secure data transmission application requires to deploy an encryption and decryption step in the source and destination domains, respectively.

We can solve this generalized problem with k processing sites ($k \geq 1$) in a way similar to the single site case. The target graph G has $k+1$ layers, each layer representing a copy of the original graph. We let v_i denote the copy of node v in layer i. For each processing node $r \in R_i$, we add an edge (r_i, r_{i+1}) in the target graph and set the cost to $c_i(r)$ from the original graph. Figure 6 illustrates an example graph transformation for $k = 2$ processing sites. Again, by projecting the target graph to the original graph the locations of the processing sites can be determined.

IV. Hierarchical, Distributed Mapping

In the previous section we explained how we can determine processing sites assuming a complete network view. However, this assumption is not realistic for large networks consisting of thousands of nodes. In this section we address scalability issues and show how our scheme can be extended to very large networks.

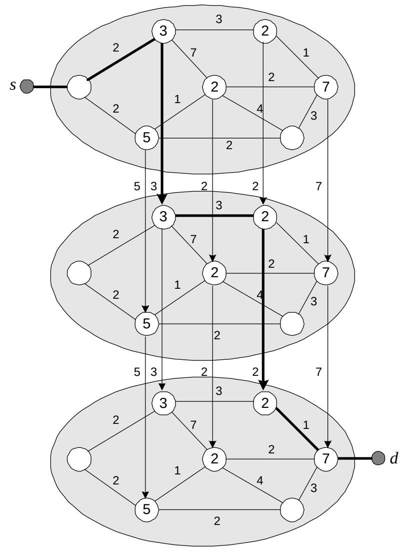

Figure 6: Graph transformation for multiple sites

A. Hierarchical Network Structure

To make our approach scalable to very large networks, we aggregate information about sections of the network, similar in nature to [1]. Nodes in the network are partitioned into groups of interconnected nodes called peer groups. A peer group appears as a single logical node at the next level of the hierarchy. Figure 7 shows a simplified example of a physical network partitioned into three peer groups and a logical hierarchy level which has been built on top.

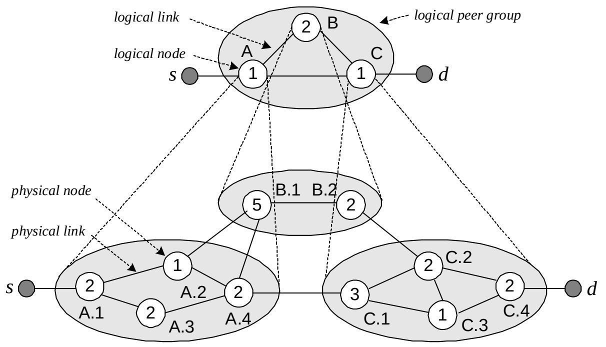

Figure 7: Hierarchical network structure

Each peer group has an assigned peer group leader that acts as a logical group node for the next hierarchy level and aggregates and distributes topology and state information to maintain the hierarchy. Apart from its specific role in summarizing and distributing peer group information, it acts like any other node.

B. Network Attribute Aggregation

We also need to summarize network attributes and propagate them to higher levels. The goal of aggregation is to provide useful information to higher levels for making appropriate decisions for selecting processing sites. How individual attributes are summarized depends on the type of the attribute and what information should be propagated up the hierarchy. Figure 8 illustrates how attributes of various nodes can be summarized. Network address ranges are aggregated using a summary address with a shorter prefix length. For processing site costs, there are several schemes possible. In an optimistic case, we could advertise the least-cost node, giving the session configuration system a hint where to find cheap processing sites. To be more conservative, we could announce the average cost of all processing sites. A node can also have other attributes that describe whether it supports a given protocol (such as RSVP) or provides a specific service (such as DNS). The RSVP attribute is only propagated up the hierarchy if *all* nodes indeed support it, thus the aggregation corresponds to a logical-AND of all RSVP attribute values. The DNS attribute is only announced if *at least one* node provides such a service within a peer group, that is, the summarization is a logical-OR of all attribute values.

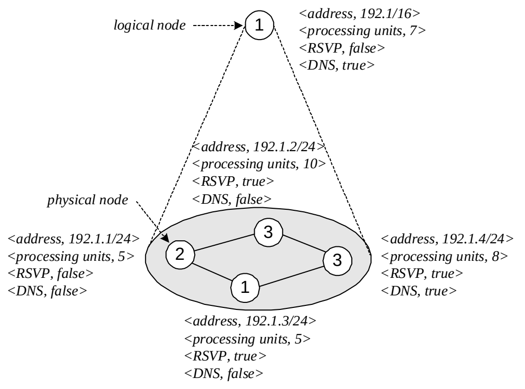

Figure 8: Aggregation of network attributes

The aggregation methods described in this paper are only a subset of possible summarization rules. Finding suitable aggregation schemes is a topic for further investigation.

C. Hierarchical Session Setup

With the introduction of network hierarchies, the session establishment needs to be performed in a hierarchical way, that is applying the algorithm described in Section III first on a higher level (using aggregated information) and then executing the same algorithm on individual peer groups at lower levels (which have more accurate information). When a session establishment request arrives at the router, that node is responsible for determining the hierarchy level that is needed to process the setup by examining the network addresses of the source, destination and intermediate processing sites. If not all nodes belong to the peer group of the current hierarchy level, the session setup request is delegated to the peer group leader which acts as a logically higher peer group node.

Figure 9 illustrates this process where the session establishment has been delegated to a higher level peer group that includes logical group nodes *A*, *B*, and *C*. The peer group leader executes the routing algorithm based on summarized information and selects a path and processing sites that seem to be capable of providing the requested computation. In the example, it selects the path (*s*, *A*, *B*, *C*, *d)* with processing steps p_1 and p_2 within *A* and computation p_3 in *C*. Since the final path needs to be a path on the physical topology, the session configuration system uses a divide and conquer

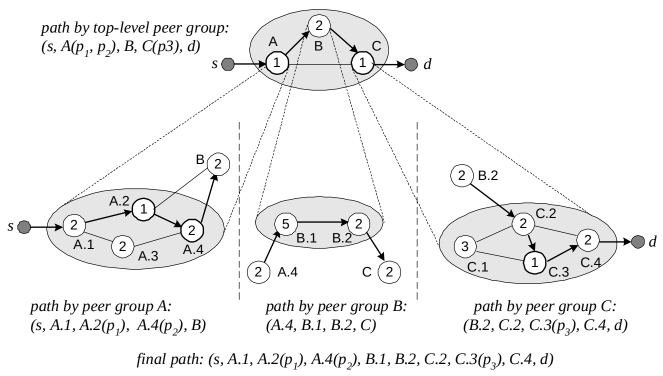
path by top-level peer group:
(s, A(p1, p2), B, C(p3), d)
s
d
A
B
C
path by peer group A:
(s, A.1, A.2(p1), A.4(p2), B)
path by peer group B:
(A.4, B.1, B.2, C)
path by peer group C:
(B.2, C.2, C.3(p3), C.4, d)
final path: (s, A.1, A.2(p1), A.4(p2), B.1, B.2, C.2, C.3(p3), C.4, d)
A.1
A.2
A.3
A.4
B.1
B.2
C.1
C.2
C.3
C.4

Figure 9: Divide and conquer approach to setup session

approach. When a path transits a logical node, it delegates the routing through that particular peer group to the lower hierarchy level.

The routing algorithm described in Section III is recursively applied to a more detailed subsection of the initial network. In the example given, peer group *A* performs path and processing site selection through *A* itself and chooses the path *A.1*, *A.2*, *A.4* with processing on *A.2* and *A.4*. The routing algorithm also selects an external path to peer group *B* that seems to be a good candidate to reach *B* based on *A*'s topology and state information. Since *A* does not have complete information about the neighboring peer group *B*, this path may not be globally optimal. However, we believe this trade-off is acceptable since optimal routing must generally be sacrificed for scalability reasons.

Once the session setup system has determined a path through peer group *A*, it initiates path determination through peer group *B*. As chosen by *A*, peer group *B* will be reached from node *A.4*. Peer group *B* runs a simple shortest-path algorithm (since no processing is required in peer group *B*) to reach *C* and selects the path *B.1, B.2*. Finally, the session establishment system initiates routing through *C*. Again, peer group *C* runs the routing algorithm within its own peer group and chooses the path *C.2*, *C.3*, *C.4* with processing on *C.3*. Finally, we have found in a distributed manner a complete path from the source to the destination that includes the required processing sites.

Once a complete path has been determined, the session setup system reserves the required resources along the path and installs active code on routers using a signaling protocol such as Beagle [4] and sets up a route using a mechanism such as Explicit Route Objects in MPLS [8]. This step confirms that the resources requested are in fact available. If they are not, then crankback occurs, which causes a new path to be computed if possible; thus the final outcome is either the establishment of a path satisfying the request, or refusal of the session setup.

V. Node Architecture

In this section, we identify the required components needed for an implementation and propose an architecture that obtains and maintains network state information required for optimal resource mapping.

The core component of our network architecture is the Active Pipe Subsystem (APS), a distributed session configuration system for flows that need processing on intermediate nodes. The APS accepts session initiation requests formulated as an active pipe from applications, maps session requirements onto network resources, and reserves resources along the path. First, the APS identifies the hierarchy level needed to process the request. If a session needs to be configured using multiple peer groups, the APS forwards the request to a higher instance as described in Section IV. Otherwise, it runs the path routing algorithm for the peer group it belongs to (and has complete topology information) and reserves resources along the predetermined path. Internally, the APS is composed of the components as shown in Figure 10:

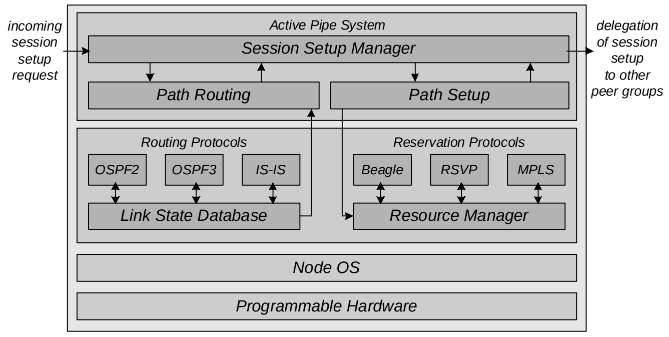

Figure 10: Node Architecture

- The **session setup manager** handles session establishment requests, determines the hierarchy level, delegates the setup to another peer group if needed or invokes routing and reservation components.
- The **path routing component** determines a route and processing sites by executing the algorithm described in Section III. To build the layered network graph, it uses topology and state information stored in the link state database. Information about processing sites can be obtained by extending routing protocols, such as using OSPF opaque fields [7].
- The **path setup component** reserves resources along a predetermined path. Using signaling protocols such as [4], active code is installed in the router's networking subsystem.

VI. Related Work

In the context of active networks, resource discovery and resource reservation are crucial factors of network programmability. Darwin [3] proposes an integrated resource management scheme where resource requests are formulated as a virtual mesh. A resource broker translates the virtual mesh onto network resources by expressing it as a boolean optimization problem, which is generally NP-hard, thus this approach is only appropriate for small to medium sized networks.

Ninja Paths [5] allow the generation of a logical path, which is a sequence of simple modules that can be composed to more complex services using operators. Ninja paths focus on composing existing services that are already installed in the network while our approach stresses optimal processing site selection and installation of new services.

End-to-end media paths [12] is a Java-based approach for building multimedia applications from components. Rules define how paths can be constructed and a pattern matching algorithm then tries to map the path onto network resources.

VII. Conclusions

In this paper, we propose active pipes as a programming abstraction that can be used by applications to specify processing requirements for active networks. An active pipe is a sequence of functions that are executed on the data stream within the network. We describe an algorithm that maps an active pipe specification onto network resources while satisfying all location constraints given by the application. We also identify components that are needed for an implementation and propose a network architecture supporting active pipes.

We believe that providing a simple yet flexible programming abstraction is important for making active applications easily programmable and widely usable. Active pipes represent a significant step towards the solution of this crucial problem.

References

[1] ATM Forum Technical Committee, *Private Network-Network Interface Specification Version 1.0*, March 1996.

[2] Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, and Daniel Villele, “A survey of programmable networks”, *Computer Communication Review*, vol. 29, no. 2, pp. 7-23, Apr. 1999.

[3] Prashant Chandra, Allan Fisher, Corey Kosak, T. S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, Hui Zhang, “Darwin: Resource Management for Value-Added Customizable Network Service”, *Sixth IEEE International Conference on Network Protocols*, October 1998.

[4] Prashant Chandra, Allan Fisher, Peter Steenkiste, “Beagle: A Resource Allocation Protocol for an Advanced Services Internet”, *Technical Report CMU-CS-98-150*, August 1998.

[5] S. Chandrasekaran, S. Madden, M. Ionescu, “Ninja Paths: An Architecture for Composing Services Over Wide Area Networks”, Computer Science Division, University of California, Berkeley.

[6] Sumi Choi, Jonathan Turner, Tilman Wolf, “Configuring Sessions in Programmable Networks”, *Proceedings of Infocom 2001*, March 2001.

[7] Rob Coltun, *RFC 2370 The OSPF Opaque LSA Option*, IETF Network Working Group, July 1998.

[8] B. Davie, T. Li, E. Rosen, Y. Rekhter, “Explicit Route Support in MPLS”, Internet Draft, November 1997.

[9] J. Moy, *RFC 2328 OSPF Version 2*, IETF Network Working Group, April 1998.

[10] David Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, Gary J. Minden, “A Survey of Active Network Research”, *IEEE Communications Magazine*, Vol. 35, no. 1, pp. 80-86, January 1997.

[11] Roch A. Guérin, “QoS Routing in Networks with Inaccurate Information: Theory and Algorithms”, *IEEE Transactions on Networking*, Vol. 7, No. 3, June 1999.

[12] Akihiro Nakao, Larry Peterson, Andy Bavier, “Constructing End-to-End Path for Playing Multimedia Objects,” Department of Computer Science, Princeton University.

[13] Konstantinos Psounis, “Active Networks: Applications, Security, Safety, and Architectures”, *IEEE Communications Surveys*, First Quarter 1999.