Configuring Sessions in Programmable Networks

Sumi Choi, Jonathan Turner and Tilman Wolf
Department of Computer Science, Campus Box 1045
Washington University, St. Louis, MO 63130-4899
{syc1, jst, wolf}@arl.wustl.edu

Abstract—**The provision of advanced computational services within networks is rapidly becoming both feasible and economical. We present a general approach to the problem of configuring application sessions that require intermediate processing by showing how the session configuration problem can be transformed to a conventional shortest path problem for unicast sessions or to a conventional Steiner tree problem for multicast sessions. We show, through a series of examples, that the method can be applied to a wide variety of different situations.**

Keywords—**routing, programmable networks, session configuration**

I. Introduction

Advances in technology are making it possible to incorporate general purpose processing capabilities in network routers. Network processor components with more than ten RISC cores have recently become available and will soon appear in high performance routers from several different equipment vendors. Research in *active networking* [1], [2], [3] is exploring the potential of programmable routers, and other approaches are being pursued by individual router vendors.

This paper is concerned with the problem of how to map application sessions onto network resources, when those network resources may include computational elements that perform some service on behalf of the application. For example, a video application might invoke a video compression service in the network to reduce its usage of network bandwidth. There may be several places in the network where the required compression and decompression service could be performed. We would like to select the best locations that meet the application's requirements. In this paper, we describe a general methodology for configuring such applications so as to make most effective use of network resources, including link bandwidth and the computational resources provided by the network. Our methodology is not restricted to systems in which application services are provided at routers. It can also be used to configure application services provided by network-attached servers.

We assume an operating environment in which application sessions are explicitly configured when the application starts up. The configuration of an application session includes selection of intermediate processing nodes and the network links used for communication among the various components of the application. In our view, this session-oriented approach is needed to enable efficient allocation of network resources among competing applications. This is especially true for applications that require a certain level of resources in order to achieve an acceptable quality of service. However, even "best-effort" applications can benefit from a resource allocation system that seeks to configure applications to take advantage of locations where resources are plentiful, rather than simply letting them compete for resources in locations where the required resources may be scarce.

Sections 2 through 7 describe various application scenarios that each raise different resource configuration issues. In each case, we show how the problem can be reformulated so that it can be solved in a similar fashion. In Section 8, we discuss implementation issues and in Section 9, discuss how it can be applied to sessions that require explicit resource reservation. We provide related works of resource allocation and configuration in Section 10 and conclude in Section 11.

II. Routing Through One Processing Site

We start with the simplest version of the application configuration problem. In this version, we have two participating end systems and there is some intermediate computation that is to be performed somewhere in the network (possibly a format translation, for example, allowing two otherwise incompatible end systems to share information). There are a number of sites within the network where the processing could occur, but not all of the sites may be able to perform the needed processing (perhaps they are not capable of executing the required program, or perhaps their computational resources are already fully committed to other tasks). The application configuration system must select one of the sites within the network and select network paths joining the end systems to the intermediate processing site. It should do this in such a way as to minimize the use of network resources, including link bandwidth and processing "bandwidth."

We can state the problem formally as follows. The network is represented by a directed graph, $G = (V, E)$, in which the nodes correspond to routers and end systems, while the edges correspond to links. Let $R \subseteq V$ be a subset of the nodes that represent sites where intermediate processing may occur. For brevity, we'll refer to these as *red* nodes. Each edge (u, v) has an associated *cost* $c(u, v)$ and each red node r has an associated cost $c(r)$. Finally, we have a source vertex s and a destination vertex t. Our objective is to find a least-cost path from s to t that includes at least one red node. The cost of a path is the sum of the costs of its links, plus the cost of the least cost red node along the path. Note that the overall path from s to t may not be a simple path. See Figure 1 for an example of the problem. The *red* nodes can be distinguished from the other nodes by the numbers that indicate their processing costs. The heavy weight edges in the figure indicate the best path from s to t that passes through at least one red node.

There is one fairly obvious approach to solving the problem. First, solve the single-source shortest path problem from s to all other nodes [8], considering link costs only. Second, solve the single-destination shortest path problem to t from all the other nodes. At the end of these two steps, for each vertex u, we know the cost of the shortest path from s to u and from u to t. So we

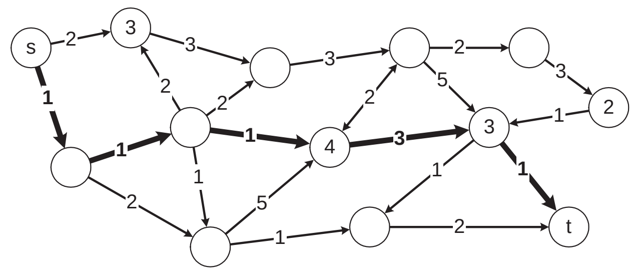

Fig. 1. Network with Processing Sites

can simply iterate over all nodes $r \in R$ and select the node that minimizes

$$d(s, r) + d(r, t) + c(r)$$

where $d(x, y)$ denotes the length of the shortest path between x and y, considering just the edge costs. For a graph with n vertices and m edges, this algorithm can be implemented to run in $O(m + n \log n)$ time. This is the same complexity as that for finding a shortest path in a graph, so we cannot expect to improve on it substantially. The only real drawback of this method is that it does not readily generalize to more complex situations. For that reason we consider an alternative approach that can be applied more generally.

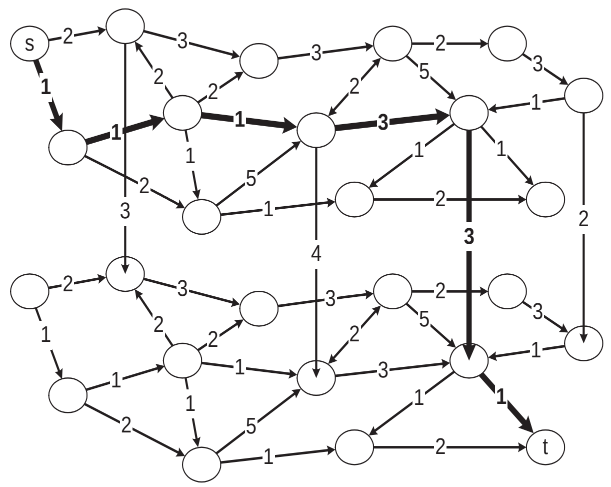

Fig. 2. Transformed Network for Single Site Processing

Our alternative approach is to transform the original problem to a conventional shortest path problem on a different graph. We then solve this new problem using standard methods and apply the results back to the original problem. The first step in the transformation is to make two copies of the original graph G. We refer to these two copies as *layers* in the resulting graph and identify them as layer 1 and layer 2. For each vertex u in the original graph, let u_1 denote the copy of u in layer 1 of the target graph and let u_2 denote the copy of u in layer 2. The edges in the two layers have the same costs as the corresponding edges in the original graph. Now, for every node $r \in R$, we add an edge (r_1, r_2) in the target graph and let $c(r_1, r_2)$ be equal to the cost originally assigned to r. This completes the construction of the target graph. See Figure 2 for an illustration of the construction. To solve our original problem, we simply find a shortest path from s_1 to t_2 in the target graph, considering link costs only (see Figure 2). The resulting path can then be mapped back to a path in the original graph by "projecting" the two layer path onto a single layer.

The correctness of this procedure is easily established. First, note that the least cost path from s_1 to t_2 does correspond to a path (not necessarily a simple path) in the original graph and the cost of the path is the same as the cost defined in the original problem statement for the corresponding path in the original graph. Second, note that there cannot be a cheaper solution to the original problem. If there were, this solution would have to correspond to a path from s_1 to t_2 in the target graph that is cheaper than the given least-cost solution, a clear contradiction.

III. Routing Through Multiple Sites

In this section, we consider a more general application configuration problem. There are again two participating end systems, but here there are several intermediate computational steps that are to be performed at possibly different locations in the network. For each step, there may be multiple sites where the processing could be done. One simple example is secure data transmission, where the intermediate processing steps include encryption and decryption processing. The encryption processing can be done at any of several nodes in the originating end system's domain and decryption processing can be done at any of several nodes in the destination end system's domain. We allow k intermediate processing steps for any $k \geq 1$.

We can state the problem formally as follows. The network is represented by a directed graph, $G = (V, E)$, with each edge (u, v) having an associated cost $c(u, v)$. As before, we have a source node s and a destination node t. For $1 \leq i \leq k$, let $R_i \subseteq V$ be a subset of the nodes. R_i contains sites where the i^{th} intermediate processing step may be performed. Accordingly, each node $r \in R_i$ has an associated cost $c_i(r)$. We define an *admissible path* from s to t to be a path (not necessarily simple) that includes nodes from each of the R_i, appearing in order. That is, a path $u_1, u_2, \ldots, u_m$ is admissible, if there are integers $i_1, \ldots, i_k$ that satisfy $1 \leq i_1 \leq \cdots \leq i_k \leq m$ and $u_{i_j} \in R_j$ for $1 \leq j \leq k$. The list of nodes $(u_{i_1}, \ldots, u_{i_k})$ is called a *site list* for the path. An admissible path may have multiple site lists. Note that a node may appear in a site list more than once. The cost of a site list is the sum of the costs of its nodes and the cost of an admissible path is the sum of the costs of its edges, plus the cost of its least expensive site list. Figure 3 shows an example of the problem. In this figure, nodes drawn with "thick" circles are in R_2, while the other nodes containing numbers are in R_1.

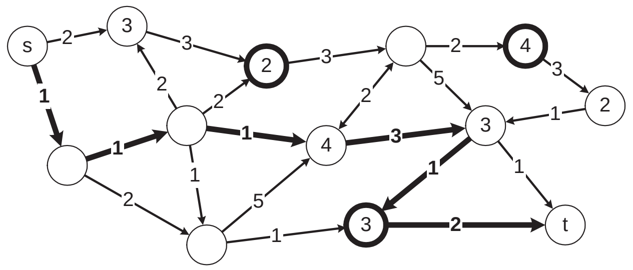

Fig. 3. Network for Multiple Site Processing

A brute force approach to solving this problem involves enumerating all possible combinations of processing nodes and connecting them with the shortest paths. However, the number of possible combinations grows proportionally to n^k, making this approach impractical, even for modest values of k.

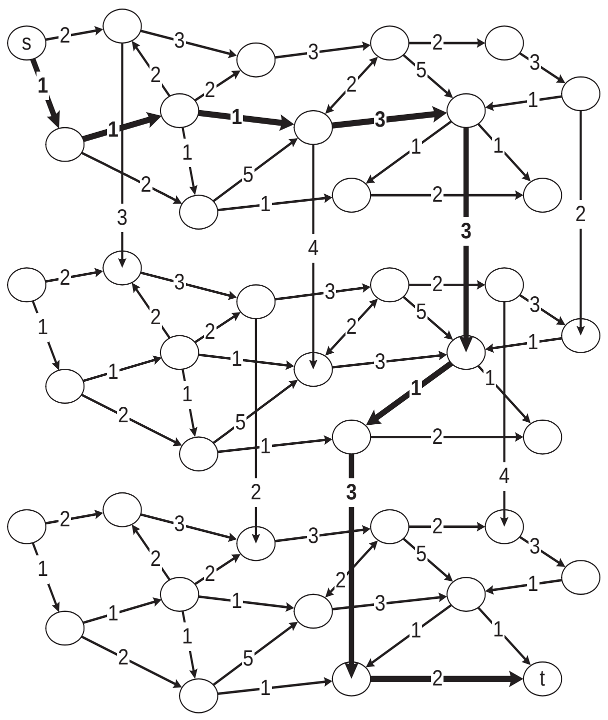

Fig. 4. Transformed Network for Multiple Site Processing

Fortunately, the problem can be solved efficiently be reducing it to an ordinary shortest path problem in a different graph. The target graph G has $k + 1$ layers, each layer being just a copy of the original graph, and numbered from 1 to $k + 1$. For each node u in the original graph, we let u_i denote the copy of u in layer i. Now, for every node $r \in R_i$, we add an edge (r_i, r_{i+1}) in the target graph and let $c(r_i, r_{i+1})$ be equal to the cost $c_i(r)$ assigned to r in the original graph. See Figure 4 for an example of a target graph for a problem with $k = 2$. To solve the original problem, we find the shortest path from s_1 to t_{k+1} in the target graph. The resulting path can be mapped back to a path on the original graph by "projecting" the path back onto the original graph.

The correctness of the procedure can be shown in a similar fashion as in Section 2. Consider the least cost path from s_1 to t_{k+1}. It is easy to see that it corresponds to an admissible path in the original graph and that its cost is the same as the cost of the admissible path. Also note that there can exist no cheaper solution to the original problem. Any cheaper solution would have to correspond to a path from s_1 to t_{k+1} in the target graph, yielding a contradiction to the definition of the shortest path.

IV. Applications that Alter Bandwidth

Certain processing steps performed on behalf of an application may alter properties of the data. For example, processing steps that compress data can change its bandwidth requirements by substantial amounts. We would like to be able to configure compression and decompression processing in the network, so as to best exploit the savings that can be obtained, while simultaneously accounting for the costs associated with the compression algorithm itself. More generally, we want to be able to configure arbitrary applications that modify the bandwidth requirements of the processed data stream. Examples for applications that decrease the bandwidth of a stream are data and image compression, filtering, and data merging. Applications that increase the bandwidth of a data stream are data and image decompression, forward error correction coding, certain encryption and authentication schemes, etc.

To quantify the changes in bandwidth, we define the *bandwidth scale factor* γ_i for processing step i, to be the ratio of the outgoing bandwidth to the incoming bandwidth for processing step i. The application configuration problem introduced in the previous section can be generalized to handle changes in bandwidth requirements. The only change needed is to the definition of the cost of an admissible path, to account for the changes in the bandwidth of the data stream. Let $P = u_1, \ldots u_m$ be an admissible path, that includes the site list $L = (u_{i_1}, \ldots u_{i_k})$. The cost of P with respect to site list L is given by

$$\begin{aligned}
&\sum_{j=1}^{i_1-1} c(u_j, u_{j+1}) + c(u_{i_1}) \\
&\quad + \sum_{j=i_1}^{i_2-1} \gamma_1 (c(u_j, u_{j+1}) + c(u_{i_2})) \\
&\quad + \sum_{j=i_2}^{i_3-1} \gamma_1\gamma_2 (c(u_j, u_{j+1}) + c(u_{i_3})) \\
&\quad + \cdots \\
&\quad + \sum_{j=i_{k-1}}^{i_k-1} (\gamma_1\gamma_2 \cdots \gamma_{k-1})(c(u_j, u_{j+1}) + c(u_{i_k})) \\
&\quad + \sum_{j=i_k}^{m-1} (\gamma_1\gamma_2 \cdots \gamma_k) c(u_j, u_{j+1})
\end{aligned}$$

The cost of a path P, is the the minimum over all site lists L of P, of the cost of P with respect to L.

The solution method of the previous section can also be generalized to handle bandwidth scaling. The target graph is constructed as before, but the edge costs of the target graph are modified as follows. For edges within layer i, the edge costs are multiplied by $\gamma_1\gamma_2 \cdots \gamma_{i-1}$. Edge costs from layer i to layer $i + 1$ are multiplied by $\gamma_1\gamma_2 \cdots \gamma_i$. We solve the problem, as before, by finding a shortest path from s_1 to t_{k+1}.

V. Optional Processing

Some network applications provide services that are not necessary for correct data transmission, but which can improve the performance or quality of the connection. These optional processing steps might decrease the transmission cost to some destination nodes, but not necessarily to all. We now extend our method to handle such cases.

For concreteness, we use a simple example of a compression/decompression application. The processing for compression and decompression incurs a cost, but the intermediate data

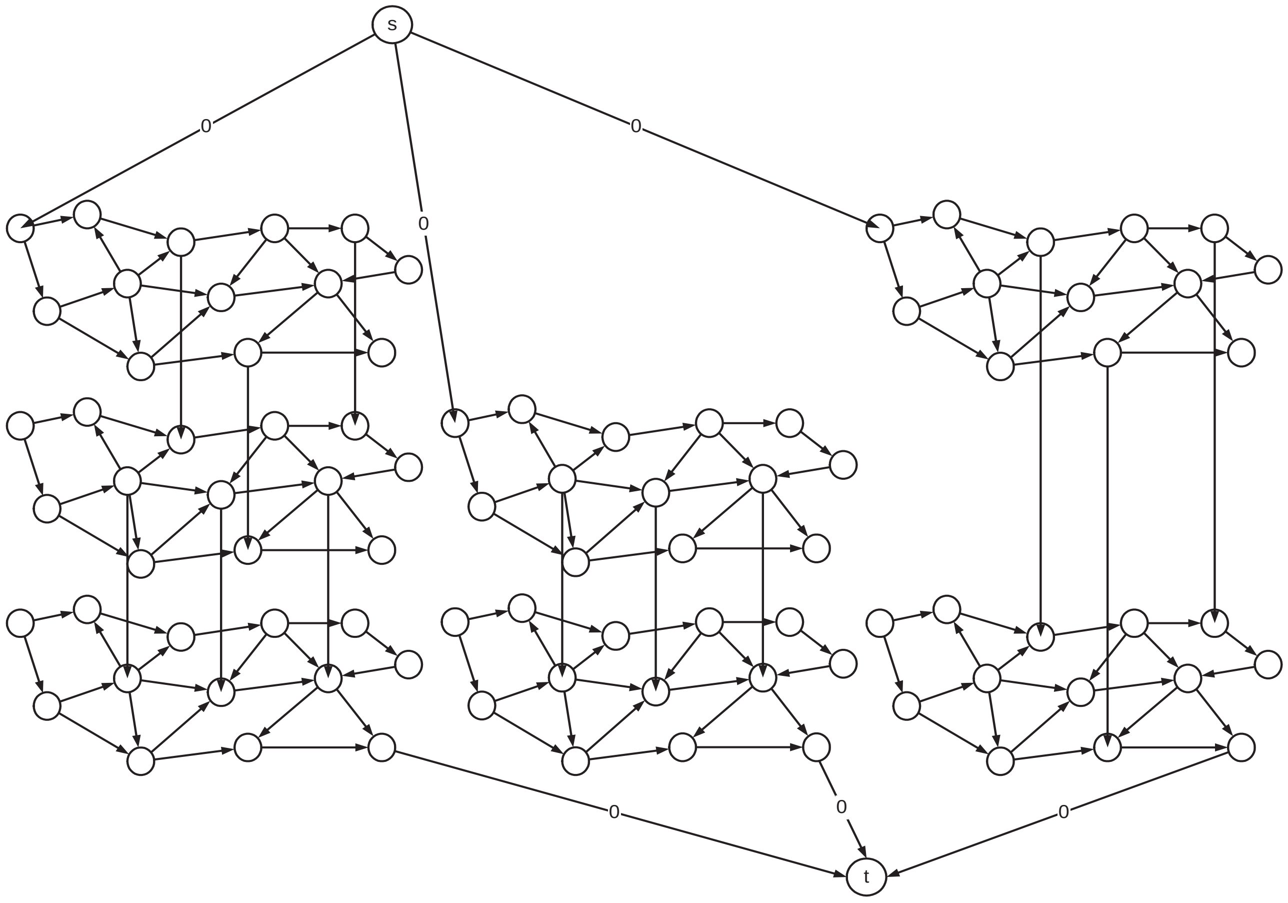

Fig. 5. Transformed Network for Optional Processing

stream has a lower bandwidth ($\gamma < 1$) which yields lower transmission costs. Thus, for long-distance transmissions the processing overhead is worthwhile, while for short distances, the cost of the added processing may exceed the benefit. The problem can be solved using the method of the previous section. To make the compression and decompression processing optional, for each vertex u in the original graph, we add edges (u_1, u_3), linking layers 1 and 3. These edges are assigned a cost of zero. Note, that for this method to work correctly, the bandwidth of the decompressed data stream must match that of the original, uncompressed data stream. In this case, we can actually use a slightly simpler target graph with just two layers, and edges (u_1, u_2) for all vertices $u \in R_1$ and edges (v_2, v_1) for all vertices $v \in R_2$. The edges within layer 2 are scaled by the compression factor, as are the edges from layer 2 to layer 1.

The method can be extended to configuring sessions where different processing stages are optional. However, when the effects on the bandwidth of the data stream are more complex than in the simple compression/decompression example, a more complex target graph may be required. These more general cases can be solved using target graphs that have a source node s connected to multiple columns of layers, where each column contains some subset of the layers for the complete processing, and eventually connected to the destination t below the last layer of each column. The general form of such a graph is illustrated in Figure 5. The columns of layers connected from the source s and to the destination t represent possible choices of processing sequences.

VI. Congestion Control Processing

Application-specific congestion control [9] is often cited as a good example application for active networking. The idea is that an application-specific module could modify the application data stream dynamically in response to network congestion, in a way that minimizes the impact on the application (for example, a video congestion control module might preferentially discard high frequency information, to reduce the subjective impact of the lost information).

For this type of application, the modules should be installed at nodes preceding those links that are most likely to be subject to congestion, but can be omitted from links where congestion is unlikely to occur. If the application is configured to use several congested links, the congestion control module will need to be installed at each of these links. If it is configured to use only uncongested links, then no congestion control modules need to be installed. If a path using several congested links is much shorter than a path that uses no congested links, it may be preferred. We want to formulate the problem so that we can make the best overall choice of a path, considering both the cost of the links and the cost associated with the congestion control (this may include both a processing cost component and a “cost” for the impact of congestion on the application). We can accomplish this simply by modifying the costs of all congested links to reflect the added cost of coping with congestion at those links, and then we search for a shortest path, using the modified costs.

The problem is defined formally as follows. The network is represented by a graph $G = (V, E)$ and we let $L \subseteq E$ denote the set of *congested* links. Each edge (u, v) in the graph has an associated cost, $c(u, v)$ and each congested edge has additional cost $c'(u, v)$. Given a source s and a destination t, our objective is to find a least-cost path from s to t. The cost of a path includes the cost of its links, where for congested links we include both c and c' in the sum.

VII. Configuring Multicast Sessions

So far, we have considered several types of different application configuration problems with two participating end system and the common objective to find an optimal path from one to the other. In this section, we show that our method can be applied to multicast applications where there are multiple destinations, rather than just one. For each of the source-destination paths, we want to include the same sort of processing that we might apply to a unicast application. Our objective is to find a way of selecting processing sites and links so that the processing requirements are met, and so that the overall cost is minimized.

We illustrate the application of the method to multicast situations by considering a video distribution application, where we need to perform compression processing and decompression processing. As discussed earlier, we can solve this problem for unicast applications using a two layer graph with "compression edges" from layer 1 to layer 2, and "decompression edges" from layer 2 to layer 1. The same target graph can be used for the multicast problem, where we have a source and multiple destinations. See Figure 6 for an example of the target graph. The only real difference is that the objective of the problem becomes finding a least-cost subtree of the two layer network with the source at the root, and the destinations at the leaves. This problem is a Steiner Tree problem (as is the usual multicast routing problem), which is known to be NP-complete [10], [11]. There are several known approximation algorithms for the Steiner Tree problem in graphs that can produce solutions costing no more than twice the cost of an optimal solution, and which in practice are typically better than the bound implied by the worst-case performance. We do not discuss such algorithms further here; we simply note that they can be applied to finding an appropriate tree in the target graph, and we can then use this to produce a solution to the original multicast session configuration problem.

VIII. Resource Reservation Issues

So far, we have not explicitly raised the issue of resource reservation. While link and node costs may be defined to express the availability of resources, there are some additional issues that must be addressed, in order to handle resource reservation correctly.

One way to handle resource reservation is to simply omit from the original network graph those links and nodes that lack sufficient resources to handle a given application session. So for example, if an application session requires 100 Mb/s of link bandwidth, we can simply remove from the network graph all

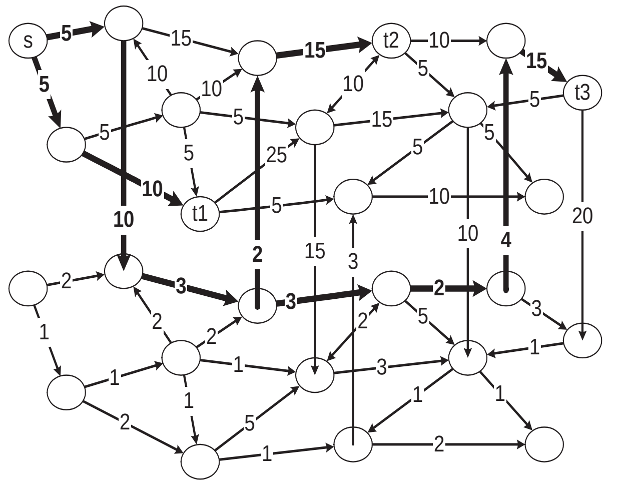

Fig. 6. Transformed Network for Multicast with Compression

edges corresponding to links with less than 100 Mb/s of available bandwidth. Similarly, if it requires the equivalent of 50 MIPS of "CPU bandwidth" at one processing site, we can omit all nodes that have less than 50 MIPS of available capacity. Unfortunately, this does not quite work, since the problem specifically allows edges and nodes to be used more than once by an application session. If our example session were to use a given link twice, that link would require 200 Mb/s of unused bandwidth, in order to accommodate both uses. Similarly, a node that is used in more than one processing step, must have sufficient available CPU bandwidth to handle all the steps for which it is used.

Unfortunately, there appears to be no simple solution to this problem. Consider an instance of the session configuration problem in which every node but the source and sink is a potential processing site, there are $n - 2$ intermediate processing steps (where n is the number of nodes in the graph, including the source and sink), but each node has only enough CPU bandwidth to perform one step. Problem instances like this correspond directly to a variant of the NP-hard traveling salesman problem, so we cannot expect to find an efficient optimal algorithm for the session configuration problem that accounts for resource reservation.

However, it appears likely that in practice, a more positive outcome can be expected. Real applications typically involve just a few processing steps, opening up opportunities for finding optimal or near-optimal solutions with modest computational effort. Specifically, our approach of transforming a session configuration problem to a shortest path problem in a layered graph can also be usefully applied to problems where there are resource reservations.

Consider the version of the problem discussed in Section III. If the session to be configured requires a link bandwidth of B and the layered graph has k layers, then omit from the layered graph all edges corresponding to links for which the available bandwidth is less than kB. Similarly, if the sum of the processing resources required at all sites is R, omit from the layered graph all inter-layer edges corresponding to processing sites with less than R units of CPU bandwidth. A shortest path in this version of the layered graph is guaranteed to correspond to a feasible solution, although possibly not an optimal solution. Assuming such a solution exists, it does at least provide an upper bound on the cost of an optimal solution. We can also obtain a lower bound on the cost of an optimal solution by including in the layered graph all intra-layer edges corresponding to links with B units of available bandwidth and all nodes with enough available CPU bandwidth to handle a single processing step. A shortest path in this version of the layered graph is guaranteed to have a cost no larger than that of an optimal solution to the session configuration problem, but the solution obtained may not correspond to a feasible solution, since it may use nodes or edges more times than resource constraints would allow. We expect that in practice, the upper and lower bound solutions obtained in this way will often differ by only small amounts, making the use of the "upper bound solution" a reasonable decision.

More realistic algorithms can be obtained by refining these methods. For example, if an intra-layer edge has iB units of capacity, but less than $(i+1)B$, it can safely be included in any i layers of the network graph. Including such edges in i layers will increase the number of feasible paths that can be discovered by a shortest path search in the layered graph, leading to more nearly optimal solutions. This can be taken a step further by adapting Dijkstra's shortest path algorithm so that it builds a shortest path tree in a layer-by-layer fashion (this works because edges are directed from one layer to the next, and not back). As we go from one layer to the next, we can decide whether to include edges in the next layer or not. Specifically, any edge with iB units of available bandwidth, which has so far been included in the shortest path tree in $< i$ layers can be safely included in the next layer without violating any resource constraint.

Evaluation of these and other heuristic approaches will have to be addressed through simulation studies of session establishment in realistic network configurations. Such studies are now in progress.

IX. Implementation Issues

The previous sections have omitted any explicit discussion of implementation strategies, focusing instead on fundamental algorithmic issues. Of course, in order to apply our approach, a suitable implementation method is needed. One way to implement the approach is for a global configuration server, to make session configuration decisions, based on complete knowledge of the network state that it maintains at a central location. While this may be feasible in small networks, it clearly does not scale to larger systems. In general terms, what is needed is a distributed configuration service, that allows configuration decisions to be made by multiple computers in a cooperative fashion. Such a system must include a component that distributes information about network resource availability, and a component that uses that information to make configuration decisions with respect to specific sessions.

The ATM *Private Network-Network Interface* protocol (PNNI) [12] is an example of a distributed resource allocation system that solves a similar problem. PNNI can be viewed as two protocols, a link-state protocol that distributes information about network resource availability, and a signalling protocol that uses this information to make virtual circuit routing decisions. In the case of PNNI, the route from a source to a destination is selected by the switch connected to the source, using stored information about the network topology and resource availability. It then passes the selected route to other switches along the path. They, in turn, make local resource reservations and propagate the signalling message along the path. If during this process, an attempt to make a local resource reservation fails, a new path may be computed by the switch at the point where the reservation failed, allowing the path setup process to continue. To make the approach scalable to very large networks, the PNNI protocol aggregates information about sections of the network, allowing switches to have complete knowledge of the portion of the network that is close to them and more summary knowledge of distant portions of the network.

The general approach taken by the PNNI protocol can be extended to handle configuration of sessions requiring intermediate processing. The state information distributed by the routing protocol must be expanded to include information about processing resources available at various locations in the network. Using this information, a path can be computed by the router connected to the source of a unicast session, and then forwarded in a signalling message to successive routers on the path to the destination, allowing local resource reservations to be made as the signalling message proceeds to the destination. Of course, as with the basic PNNI protocols, the selected paths may not be globally optimal, since initial path selections may be based on summary information about distant portions of the network. This is nothing new in network routing, where optimality of path selection must generally be sacrificed for the sake of scalability.

Other approaches are possible as well. In particular, other link state protocols, such as Open Shortest Path First (OSPF) [13] can be used to distribute state information, and other signalling protocols can be used to select paths and make the required resource reservations.

X. Related Work

Resource discovery and resource allocation are important elements of network programmability. The Darwin project [4] proposes a set of resource management mechanisms that support customized network services. Their resource management system is divided into four components, high-level resource allocation, run-time resource management, hierarchical resource scheduling and low-level resource allocation. A *service broker* component called Xena provides both resource discovery and allocation. Xena formulates the resource allocation problem as a general optimization problem with multiple metrics. While this provide a very flexible and general formulation, it makes it computationally infeasible to find optimal solutions, even in simple situations. By contrast, our approach sacrifices some degree of flexibility to enable rapid computation of optimal solutions in the most common cases.

A different approach is taken in [5], where the focus is on identifying topological properties related to network services and resource states. Constrained programmability is provided to applications based on these properties. Topological properties are determined by distributing network queries and then aggregating results back at the source, using a form of network fusion operation.

Market-based resource control mechanisms are considered in [7]. In this work, resources are treated as trade goods, network nodes and links as producers and applications as consumers. Service brokers are used to mediate access to resources between producers and consumers, using a form of currency exchange, and can enable varying levels of competition and cooperation.

XI. Summary

The provision of advanced computational services within networks is rapidly becoming both feasible and economical. The provision of such services, either by routers or by network-attached processing sites, is potentially a significant benefit for network users, as it can relieve individuals from the need to acquire, install and maintain software in end systems to perform required services. As such network services become more widely used, it will become increasingly important for service providers to have effective methods for configuring applications sessions so that they use resources efficiently.

We have presented a general approach to the problem of configuring application sessions that require intermediate processing. The method involves transformation of the original problem to a conventional shortest path problem. We have shown, through a series of examples, that the method can be applied to a wide variety of different situations. To make the ideas in this paper directly applicable, it will be necessary to automate the methodology, so that resource management software can automatically determine the best way to configure a session to satisfy its requirements. The next step in reaching this objective is to develop a general way of specifying application requirements for intermediate processing, that is expressive enough to describe typical application scenarios, while being simple enough for application programmers to use effectively.

We believe that given such a specification method, it will be possible for network resource management software to combine information about network resource availability and an application specification, to produce a graph that represents the possible configurations of the application. By solving the appropriate optimization problem on this graph (typically a shortest path problem), the network resource management software will be able to automatically map the application to an appropriate set of resources. This paper represents a crucial first step in a research program that aims to achieve this objective.

References

[1] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, Jan. 1997.

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

[3] Daniel Decasper, Guru Parulkar, Sumi Choi, John DeHart, Tilman Wolf, and Bernard Plattner, "A scalable, high performance active network node," *IEEE Network*, January/February 1999.

[4] Prashant Chandra, Allan Fisher, Corey Kosak, T. S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, and Hui Zhang, "Darwin: Resource management for value-added customizable network service," *Sixth IEEE International Conference on Network Protocols*, October 1998.

[5] Youngsu Chae, Shashi Merugu, Ellen Zegura, and Samrat Bhattacharjee, "Exposing the network: Support for topology sensitive applications," *Proceedings of IEEE OpenArch 2000*, March 2000.

[6] Tomasz Imielinski and Julio C. Navas, "Gps-based geographic addressing, routing, and resource discovery," *Comm. of ACM*, vol. 42, Apr 1999.

[7] Kostas G. Anagnostakis, Michael W. Hicks, Sotiris Ioannidis, Angelos D. Keromytis, and Jonathan M. Smith, "Scalable resource control in active networks," *The Second International Working Conference on Active Networks*, October 2000.

[8] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Company, 1990.

[9] Ralph Keller, Sumi Choi, Dan Decasper, Marcel Dasen, George Fankhauser, and Bernhard Plattner, "Active router architecture for multicast video distribution," *Proceedings of IEEE Infocom 2000*, March 2000.

[10] Frank K. Hwang, Dana S. Richards, and Pawel Winter, *The Steiner Tree Problem*, vol. 53 of *Annals of Discrete Mathematics*, North-Holland, Amsterdam, Netherlands, 1992.

[11] Pawel Winter, "The steiner problem in networks: A survey.," *Networks*, vol. 17, 1987.

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

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

[14] Larry L. Peterson and Bruce S. Davie, "Computer networks," *Computer Networks : A Systems Approach, 2nd Edition*, 2000.