Automated Planning for Open Architectures

Peter Reiher, Richard Guy, Mark Yarvis, and Alexey Rudenko
University of California, Los Angeles

***Abstract* – End-to-end connections experience a high level of diversity in network characteristics. At one extreme, an application may receive highly degraded service, leaving the application unusable. At another extreme, the costs to guarantee some level of service may be undesirably high to the user or overall system. Open architecture networks help applications push adaptation technology into the network and reduce the effects of poor network characteristics. Automated planning is important for the services that are supported by the open architecture. The remedies that modify an application's data stream to adjust it to unfavorable network conditions should be located and ordered to provide good data transfer and network resources use. The search for good plans is a hard AI problem and requires additional research.**

***Index Terms*—adaptation, active networks, planning.**

I. Introduction

Some open architecture systems assume that applications must be written or re-written to take advantage of the new networking features they offer [3, 4, 7, 8]. Other open architecture systems seek to provide their benefits even to programs and data streams that are unaware of the new possibilities. Some examples of the latter are protocol boosters [5], the Berkeley proxy system [2], and Conductor [9]. These application-unaware systems sometimes require explicit user or system administrator configuration, such as designating a proxy point, or pre-deploying various forms of adaptation modules. However, this approach limits their utility, since they provide benefit only when some person is intelligent and knowledgeable enough to foresee possible benefits and take appropriate action. Another approach is to automatically apply adaptations to data streams without explicit user intervention. At a limited level, this approach is already taken by protocols such as TCP, which do not demand that human users or applications assist it in adjusting to congestion on the line. In general, automatic application of adaptations requires intelligent planning to ensure that proper, compatible adaptations are applied in appropriate places. This paper describes the problem in more detail and outlines some basic approach to a solution.

In Section II we describe the problems of automated adaptation by open architecture systems. Section III covers the Panda system, which provides support for applications that are unaware of the existence of the adaptation agent service. Section IV describes principles of automated planning of adaptations and the plan optimization problem. Section V outlines our plan for future work. Section VI provides concluding remarks.

This work was partially supported by the Defense Advanced Research Projects Agency under contract DABT63-94-C-0080. Authors may be contacted at: {reiher, rguy, yarvis, arudenko}@fmg.cs.ucla.edu.

II. The Problems of Automated Adaptation

If users are not directly involved in choosing and deploying the open architecture adaptations that a data flow needs, the system must automatically solve several problems.

First, the system must understand the format of the data stream it seeks to improve well enough to take proper actions. In some cases, not only the data stream must be considered, but the application end points of the stream, the hardware devices at those endpoints, or even the wishes and needs of the users.

Second, the system must detect problematic conditions reliably and quickly, so it can know what remedial actions to take. It must also get some sense of the longer outlook for network conditions, at least for the life of the data flow.

Third, the system must be able to apply multiple remedial actions to the same data stream. The stream may encounter multiple problems at various points along the transmission path, and generally different actions will be required to solve those problems. Applying multiple actions implies that the system must be able to determine if a set of actions are compatible. The canonical example of incompatibility is to meet problems of security and inadequate bandwidth by first encrypting the data stream, then ineffectually compressing the encrypted version of the stream.

Fourth, the system must be able to determine if the open architecture is willing and able to run all adaptations that the system proposes at the locations it chooses. Some adaptations might not run properly on particular nodes. Some nodes might be unwilling to run particular kinds of adaptations. Perhaps some nodes limit the resources expendable on a single packet or entire data flow at that node. These constraints may cause a different set of adaptations to be performed, or may affect where adaptations are located.

These last two problems require the system to be able to plan. Given knowledge of what the data stream needs to do, what problems it faces, and what possible remedial actions are available, the adaptation system must create a plan of which actions to use, in which order, located at which adaptation points in the open architecture. If there are many possible types of data streams, many possible types of problems, many possible remedial actions, many adaptation points available, and many constraints on what adaptations may be performed, creating such a plan can be challenging. Further, the plan must be created relatively quickly, since the data stream cannot be delayed indefinitely in search of the perfect plan. Depending on the specifics of the data stream, between microseconds and very small numbers of seconds are available to plan the remedial strategy. If the code

implementing remedial actions is not ubiquitous, deployment costs must also be considered in planning.

III. PANDA

We are investigating methods of automatically planning the choice and deployment of adaptations in open architectures. We are working in the context of an application-unaware active network support system called Panda. Panda automatically traps non-active data streams and converts them into streams of active packets. Panda also creates plans for which active services should be performed at each active network node or switch along the path. A Panda prototype has been used in our lab for over a year. The planning capabilities of the original prototype were extremely primitive. We are currently implementing an improved prototype that will offer better planning support.

Active services in Panda are implemented as adapters that should be deployed according to a plan on nodes designated by the plan. After the plan is activated the adapters modify all data packets arriving at the node. Adapters increase the cost of the connection, using resources such as CPU cycles, storage, network controls, etc. The deployment of adapters also requires extra time and network bandwidth.

Two adapters may have different characteristics even if they do the same kind of adaptation for a data stream. For example one adapter might compress a data stream by converting color images to black-and-white images; another adapter can achieve the same level of compression by reducing the resolution. The choice of a particular adapter in this case depends on the requirements of the user for the data stream.

The location of an adapter affects the characteristics of data stream. It might be better to extend the number of links covered by some adapters. For example, assume that some link requires the adapter that uses the Ziv Lempel technique for data compression. Then locating as many links as possible between the nodes that run compression and decompression will improve overall communication because each link will benefit by forwarding compressed data, even if it has sufficient bandwidth. At the same time it is undesirable to extend the effects of a forward error correction adapter on links that do not actually require extra reliability because this adapter increases the amount of data that must be sent.

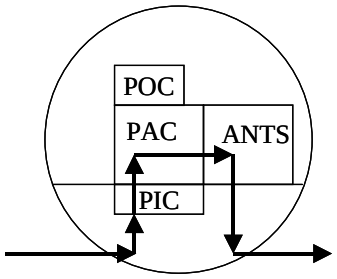

Figure 1. A Panda Node

The Panda prototype consists of three basic components (Fig. 1). The Panda Interception Component (PIC) traps messages sent by applications that do not use active networking capabilities. It examines such messages and gives those it thinks Panda can assist to the Panda Adaptation Component (PAC). The PAC is responsible for planning which adapters to use on behalf of a given data flow and deploying them at the proper locations in the network. The planning function of the PAC requires information about conditions in the network. The third Panda component, the Panda Observation Component (POC) provides this information. The POC observes network and node conditions and provides information to the PAC as required for planning. If conditions change drastically, the POC can signal the PAC, which may choose to abandon the existing plan and re-plan.

Panda uses the ANTS Execution Environment [8] to provide basic active networks services.

Panda must be deployed at any node where adapters are to be run. Panda planning requires information and cooperation from all Panda nodes traversed by a data flow.

IV. THOUGHTS ON PLANNING

The basic problem in automatic planning is to find remedies to a given set of problems located at particular nodes or links in a data path. Fig. 2 shows a simple example. Here, in a data path that traverses four links, the entire path lacks the security required for the data transmission. Also, the second link does not have sufficient bandwidth for the transmission. The third link is noisy. The final link is subject to frequent bursty cross traffic, which will tend to cause unacceptable jitter in the delivery of the data stream.

Fig. 3 shows how an open architecture system like Panda might handle the problem. Adapters (indicated as boxes) are deployed at various locations. Data encryption is performed end-to-end. Since one of the links has insufficient bandwidth, data sent across that link should be compressed; but because encryption is being done end-to-end, compression must also be done end-to-end, before the encryption is applied. The system can attach error correction codes to the encrypted data crossing the noisy link, and can use active services (such as JRSVP [1]) to reserve bandwidth on the final link. Other sets of adaptations and locations could be chosen, of course. For example, bandwidth could be reserved end-to-end, as well, or the data could be decrypted on one side of the low bandwidth link, compressed, and re-

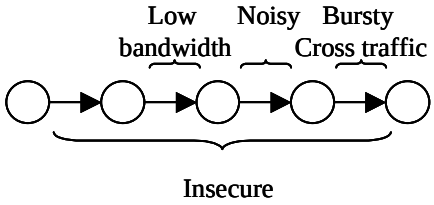

Figure 2. A data path with multiple problems

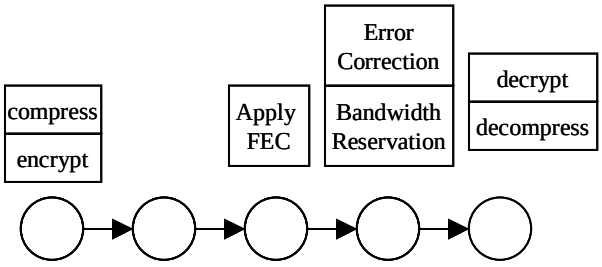

Figure 3. Adaptations deployed to remedy problems

encrypted before transmission, with reverse operations on the other side.

The problem for the planner is to go from the set of problem conditions shown in Fig. 2 and a set of possible remedies to a feasible solution, such as the one shown in Fig. 3.

One simple solution is to precompute a set of reusable plans suitable for common circumstances. If the situation seen in Fig. 2 happens frequently, someone can, in advance, devise a sensible plan for improving the situation and encode it in some format. The planner need merely find a match for its observed problems from among the set of precomputed plans. This solution has the advantage of requiring a very unsophisticated planning component, but the disadvantage of little flexibility. It can only deal with specific sets of problems we foresee. The basic idea can be extended to allow precomputed plans with slots to be filled in by the planner at runtime [6], or by allowing the planner some flexibility in where to locate adapters. The more powerful the extensions, the greater the flexibility, but the greater the complexity of the planner.

Looked at another way, finding a good plan is a searching problem. Remedies solving particular problems must be found, they must be located on nodes that are properly positioned in the data stream and that offer sufficient resources. There are a finite number of problems, a finite number of remedies, a finite number of potential remedy locations, and a finite number of constraints on what can be done at a given location for each flow. The combination of these factors defines the space of all possible plans for each flow that can be calculated. Feasible plans, the plans that can actually work for the connection, are an interesting subset. Some feasible plans are closer to optimal than the others in terms of the efficiency of the data transfer and the resources needed to deploy and run the adaptations.

One can imagine functions that define the value of certain solutions, based on whether they solve the problems faced and the costs they incur in doing so. The evaluation function must calculate a certain numerical interpretation of all factors of network communication and adapter deployment, such as throughput. Monetary cost of the use of the links in the connection is another important factor in the evaluation function. The evaluation function must also take into account the execution resources of the nodes that run adaptations, as well as the cost of deploying adapter code. A search algorithm could evaluate various possibilities to find the optimal solution, or at least a feasible solution that solves all problems at an acceptable cost.

There is an obvious tension between providing optimal behavior for a single flow and providing overall optimal network behavior. We do not presume to offer fresh insight on this problem, but suggest it can be limited by the commonly chosen means in active network research, limiting the resources devoted to a given flow network wide. Additional research in this area is ongoing in the active network community, and we will leverage this work.

We consider the number of nodes that run the adaptations as an important factor of the search strategy. The fewer the nodes running the adaptations, the smaller the solution space is, and the easier it is to find the optimal plan. However, the set of the plans that are built on a smaller number of nodes may not contain feasible plans due to a lack of resources. For instance, if only the endpoints are considered, a PDA at one endpoint may have insufficient memory or CPU cycles to adapt a video stream in real-time. Also, plans built on a limited number of nodes may not be feasible because they do not include a particular node required by an adapter. For example, the adapter that provides an electronic signature must be applied exactly at the node whose electronic signature is needed. If it is located at another node, it cannot provide the proper signature. At the same time, if all nodes are to be considered some other strategy for reducing the search space is needed.

Looking at the problem as an example of search suggests some solutions. The most obvious is an exhaustive search. If an evaluation function properly values the costs and benefits of applying various candidate solutions, exhaustive search will find the optimal solution. If the number of candidate solutions is small enough, an exhaustive search is a fine method. Consider, however, a data stream like that in Fig. 2 that faces four problems, with 256 different adapters available. Assuming one adapter is required to solve each problem, and a purely exhaustive approach to deciding which adapters to use, the system must examine over 4 billion possibilities. If the problem of adapter ordering is considered, or the problem of where to locate adapters is added, the possibilities grow.

One quick way to limit the growth of the search space is to encode adapters with the problems they solve. Instead of blindly trying all possible adapters for all possible problems, the planner can consider only adapters known to solve the particular problems being faced. In the example above, if the 256 adapters each solve a different problem, the only issues an exhaustive planner would face would be ordering the adapters and locating them on particular nodes. For a small enough number of adaptation locations, the total number of possible solutions would be reasonable. But if there are 25 different data compressors, 12 encryptors, half a dozen error-correcting encoders, and three or four reservation schemes, the number of possible solutions skyrockets. Part of the promise of open architectures is that they will allow a proliferation of adapters that help data streams, so designing a

system suitable for only a small number of adapters seems short sighted.

Non-exhaustive search strategies can start from an initial candidate solution and attempt to move towards a better solution. One initial solution is to do nothing, with each search step being the addition or replacement or relocation of an already chosen adapter. A different initial solution is to deploy some remedial adapter in the immediate vicinity of each problem. Each search step would be to replace an adapter, move it to a different location, or merge it with similar adaptors deployed elsewhere (as, for example, merging two similar compression adaptors initially located on different links). Another initial solution is to locate all required remedial adapters on the source and destination nodes, with each step relocating adapters to more appropriate locations.

The amount of work done to find superior solutions must be limited by the amount of latency acceptable to the user. If the planning process takes too long, simply sending unaltered data over an unassisted path may be better. However, if one of the as-yet-undetected problems turns out to be insufficient security, this decision could be disastrous.

V. Future Work

A number of other issues remain to be addressed. Most important, we must decide on a basic search strategy and define suitable cost functions to use for that search. Once this step is taken, we can test the search system with various alternatives, starting with simple ones and progressing to more complicated cases. When we are satisfied with the basic planning capability, other problems can be addressed.

Most open architectures work under the assumption that the underlying network is dynamic in various ways. If the network conditions change sufficiently during the course of a data connection, the existing plan may be unhelpful, or even detrimental. In such circumstances, the system should detect the problem and replan. Replanning requires other capabilities, such as the ability to cleanly switch between one set of adaptations and another. The ability to replan might suggest a strategy for plan creation in which an initial sub-optimal plan is gradually refined into a better plan while the data flows. If, however, the set of adapters used for a data flow changes very often during its life, the costs of ensuring synchronization between the different generations of plans can be high. This suggests that flexibility may be another factor in evaluating a plan.

Another interesting issue is planning for non-linear data streams, particularly multicasts. In these cases, the needs of different receivers may conflict. The planner must be able to consider tradeoffs between minimizing the number of adaptations performed (since each costs time and other resources) and providing each user with the best possible data stream.

We intend to address these issues in future versions of Panda.

VI. Conclusion

Panda is the example of an open architecture system that provides an adaptation service for end-to-end connections and automated planning for the selection and deployment of adaptations. The planning process presumes that a search for a feasible plan in the space of all possible plans will succeed. The complexity of the search depends on the scale of the plan space. We believe that in a practical system the plan space is very large making automated planning a hard artificial intelligence problem. Our general strategy will be to start with a planning system that does well for relatively simple and obvious cases, using equally simple and obvious methods. We will test and refine our planning algorithms as we gain more experience with the system.

Finding a feasible plan is limited by the temporal constraints of a real-time application. Our work is focused on the methods of fast and efficient plan space traversal for possible solutions. We have outlined what we believe to be the key components and tradeoffs to the problem. We are currently investigating which planning strategy best suits the open architecture we are working with and the kinds of problems we are interested in solving. We will soon build a planner for use in the Panda prototype.

References

[1] Bob Braden et al, Internal ISI memo, http://www.isi.edu/active-signal/ARP/ index.html.

[2] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier. "Cluster-Based Scalable Network Services", *Proceedings of the Sixteenth Intl. Symposium on Operating Systems Principles (SOSP-16)*, St.-Malo, France, October 1997.

[3] Hicks, M.; Keromytis, A.D. (Edited by: Covaci, S.) **"**A Secure Plan. Active Networks", *First International Working Conference*, IWAN'99. Proceedings, Berlin, Germany: Springer-Verlag, 1999. p.307-14.

[4] Anthony D. Joseph, Joshua A. Tauber, and Frans Kaashock, "Building Reliable Mobile-Aware Applications Using the Rower Toolkit", *Proceedings of the second ACM International Conference on Mobile Computing and Networking (MobiCom '96), Nov. 1996.*

[5] A. Mallet, J. Chung, and J. Smith, "Operating System Support for Protocol Boosters," HIPPARCH Workshop, June 1997.

[6] S.Merugu, S.Bhattacharjee, Y.Chae, M.Sanders, K.Calvert and E.Zegura, "Bowman and CANEs: Implementation of an Active Network", presented as an invited paper at the *37th annual Allerton Conference on Communication*, Control and Computing, Monticello, Illinois, September 1999.

[7] Noble, B.D., Satyanarayanan, M., Narayanan, D., Tilton, J.E., Flinn, J., Walker, K.R. **"**Agile application-aware adaptation for mobility", Operating Systems Review, vol.31, (no.5), ACM, Dec. 1997. p.276-87.

[8] Wetherall, D. **"**Active network vision and reality: lessons from a capsule-based system**",** Operating Systems Review, vol.33, (no.5), ACM, Dec. 1999. p.64-79.

[9] M. Yarvis, P. Reiher, and G. Popek, "Conductor: A framework for distributed adaptation", *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*. IEEE Computer Society Press, March 1999.