Chapter 7

Related Work

In this section we will present a number of papers that provide a view of current state of research on open network architectures and active networks, AI planning approaches, and ONA planner implementations.

7.1 ONA Implementations for ONA-Aware Applications

Some ONA technologies provide service to user applications that are aware of ONA capabilities. These systems require applications to do their own planning. Active networks [Tennenhouse96] allow users to inject customized programs into the nodes of the network. [Wetherall99] presents ANTS, a Java-based toolkit for constructing active networks. The transfer of adaptation code and the transfer of data are coupled in ANTS as in-band functions. ANTS limits the distribution of code to where it is needed, while adapting to node and connectivity failures. It improves startup performance and facilitates short-lived protocols by overlapping code distribution with execution. It allows customized processing to be expressed at a better granularity than the modification of every node in the network with all possible protocols. The adaptation code is cached in a node for all subsequent data packets to the end of the data transfer. ANTS presumes

that applications are written specifically to use it. Legacy application packets will be treated as normal IP packets without active network service.

Active networks have improved in recent years. Secure active networks [Murphy01] were developed incorporating a number of protocols for authentication, certificate and key distribution, and security policy. Thus, SANTS (Secure ANTS) uses X.509v3 for certificates, the DNSSEC protocol for certificate and key distribution, Keynote for security policy, and Java security extensions for sandboxing and security policy enforcement. However, the cost of security measures and their effect on the efficiency of packet transfer remains a major concern.

Active networks were also developed to address the resource management problem. The active resource protocol, an RSVP-like protocol, was developed at USC ISI [Braden01]. The active virtual network management protocol (AVNMP) is presented in [Bush99]. An optimistic event discrete simulation method, coupled with AN, allows optimistic prediction of the resources that will be used. The system is able to adjust the predictions that were inaccurate.

SwitchWare [Hicks99] is another example of an active network. The SwitchWare active network architecture uses three layers: active packets which contain mobile programs that replace traditional packets, active extensions which provide services on network elements and can be dynamically loaded, and a secure active network active router infrastructure which forms a high-integrity base upon which the security of the other layers depends. This security depends on integrity checking, cryptography, and verification techniques from programming languages. The authors of the architecture

also designed a special language called PLAN for protocol design. Again, the basic system assumes applications explicitly invoke its services.

[Merigu99] presents an active network comprised of the CANEs execution environment and Bowman NodeOS. Bowman is constructed by layering active network services on an existing operating system. The host operating system provides low-level mechanisms; Bowman provides a channel communication abstraction, an a-flow computation abstraction and a state-store memory abstraction, along with an extension mechanism to enrich the functionality. The CANEs execution environment provides a composition framework for active services based on customizing a generic underlying program by injecting code to run in specific points called *slots*. Again, applications must explicitly invoke CANEs services.

[Noble97] presents Odyssey, an application-aware adaptation as a collaborative partnership between operating system and applications. Odessey incorporates type-awareness for a data stream via specialized code components called wardens. To fully support a new data type, an appropriate warden has to be written and incorporated into Odyssey at each client. The wardens are subordinate to a type-independent component called the viceroy, which is responsible for central resource management.

[Joseph96] presents the Rover toolkit, which combines relocatable dynamic objects and queued remote procedure calls to provide services for mobile applications. A relocatable dynamic object is an object with a well-defined interface that can be dynamically loaded into a client computer from a server computer to reduce client/server communication requirements. A queued remote procedure call is a communication

system that permits applications to continue to make non-blocking remote procedure call requests even when a host is disconnected, with requests and responses being exchanged upon network reconnection.

7.2 ONA Implementation for ONA-Unaware Applications

Other open architecture systems seek to also provide their benefits to programs and data streams that are unaware of the new possibilities. These application-unaware systems sometimes require explicit user or system administrator configuration, such as designating a proxy point, or predeploying various forms of adaptation modules.

The execution environments of ANTS, SwitchWare, and CANEs, the viceroy of Odyssey, and the applications that are designed using the Rover kit do not contain a planning tool, as an integrated part, that selects and orders their services. Users must perform their own planning, typically at application design time.

Other open architecture systems seek to also provide their benefits to programs and data streams that are unaware of the new possibilities. 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, that do not demand that human users or applications assist in adjusting to congestion on the line.

Protocol boosters [Mallet97] are software or hardware modules that transparently improve protocol performance. The booster can reside anywhere in the network or end systems, and may operate independently, or in cooperation with other protocol boosters. Implementation of boosters requires the dynamic insertion of protocol elements into a protocol graph. In practice, protocol graphs are implemented as executable modules that cooperate via messages or shared state. Booster support requires inserting and removing the booster's function from the execution path followed for a group of packets handled by the protocol. As applications do not need to invoke protocol boosters explicitly, applications can be unaware of system services.

The Berkeley proxy system [Fox97, Fox98] offers on-demand distillation that both increases quality of service for a client and reduces end-to-end latency perceived by the client. The system consists of three main components. First, the proxy is a controller process located logically between the client and the server. In a heterogeneous network environment, the proxy should be placed near the boundary between strong and weak connectivity, e.g., at the base station of the wireless mobile network. The proxy's role is to retrieve content from Internet servers on the client's behalf, determine the high-level types of various components (e.g., images, text runs), and determine which distillation engines must be employed. Second, datatype-specific distillers are long-lived processes that are controlled by proxies and perform distillation and refinement on behalf of one or more clients. Third, the network connection monitor determines and handles the

characteristics of the client's network connection, which are the superposition of user preferences, network profile, and automatically-tracked values of effective bandwidth, roundtrip latency, and probability of packet error. Applications do not need to invoke proxy services to benefit from them.

[Liuljeberg96] presents a set of enhanced services supporting the WWW, implemented as the Mowgli Agent, Mowgli Proxy, and Mowgli Data Channel Service. The most important features of Mowgli include more efficient protocols over the wireless medium, intelligent reduction of transmitted data, background transfers reducing the burstiness of traffic, and disconnected-mode support in the form of versatile user control over caching and cellular call setup. The system provides three primary ways to reduce the transfer volume over the wireless link: data compression, caching, and intelligent filtering. Mowgli serves only WWW connections, which reduces the variety of services that are necessary to support communications. A relatively small set of pre-computed plans will easily cover all necessary cases.

Conductor [Yarvis99A, Yarvis99B] demonstrates an approach toward selecting an appropriate set of adaptive agents and a plan for their deployment. Conductor allows arbitrary adaptations to be performed along the data path without reducing the reliability of the overall system. It includes a framework and a set of protocols for deploying adapter modules into a network. The system is fully transparent to applications, allowing easy addition of new applications and new network technologies. Conductor employs a unique reliability mechanism that allows a data steam to be arbitrarily adapted at multiple points, without compromising reliability [Yarvis00].

7.2 AI Planning

Planner design for ONA remains a barely explored area. However, planning is a well-known area in artificial intelligence and operational research.

[Dean94] and [Russel95] discuss different search strategies. The simplest way to build a planner is to cast the planning problem as a search through the space of world states. Each node on the graph of possible states denotes some state of the world, and arcs connect worlds that can be reached by execution of a single action. As an improvement, the search through plan space was presented. In the graph that describes the plan space, nodes represent partially ordered plans and edges denote plan refinement operations. Partial order planning, as a refinement search within a solution plan space, is presented in [Weld94], [Kamphampati94], and [Ihrig96].

While classical planning has driven the majority of research in planning, more recently considerable attention has also been paid to planning in environments that are stochastic, dynamic and partially observable. To handle partially observable environments, information gathering is made part of the planning activity, and the classical planning techniques are extended to allow interleaving of planning and scheduling. Similarly, stochastic environments are modeled through Markov decision processes (MDP), and planning in such environments involves constructing policies for the corresponding MDPs. [Kaebling95, 98] and [Hauskrecht00] present techniques from operations research to bear on the problem of choosing optimal actions in partially observable stochastic domains. They introduce the theory of Markov decision processes (MDP) and partially observable Markov decision process (POMDP).

[Ling97] and [Bretthauer95] present a planning approach based on constrained resource planning (CRP), which is a powerful tool for solving planning and scheduling problems using a resource management focus. For example, in [Bretthauer95] a manufacturing system is modeled as an open network of queues and an optimization framework for capacity planning over a multi-period planning horizon is presented. The decision variables are the service rates (capacity) at each workstation in each time period. Capacity can be controlled at a work stations via the number of machines, modernizing or updating equipment, additional maintenance, number of workers, number of shifts, use of the overtime, etc. The model involves the minimization of capacity expansion costs or the sum of product lead times to budget constraints on capacity costs.

In [Gero98], [Jo98], and [East99], the application of genetic-engineering-based extensions to genetic algorithms the layout planning problem is presented. Genetic algorithms (GAs) are search methods inspired by natural genetics. The basic idea is founded on natural adaptive systems, where organisms evolve through generations to adapt themselves to a given environment. Recent work on genetic algorithms has demonstrated their success in solving optimization problems, showing their simple but powerful search capability. Based on the advantage of GAs, genetic evolutionary concepts have been applied to the space layout planning and have shown promising results. GA approach was further developed in [Zhou97]. Evolutionary computation (EC), developed on the basis of GA, adopts natural coding such as float point or permutation naturally to represent real-world problems and evolves them toward the

optimal solution combined with the genetic operations. This approach was widely and successfully applied in a variety of research areas.

A somewhat different vision of the problem is presented in [Kelly88]. This paper considers the question of how calls should be routed or capacity allocated in a circuit-switched network so as to optimize the performance of the network using a simplified analytical model of a circuit-switched network. The page shows the existence of implicit shadow prices associated with each route and with each link of the network, and that the equations defining these prices have a local or decentralized character. It illustrates how these results can be used as the basis for a decentralized adaptive routing scheme, responsive to changes in the demands placed on the network.

Although, we chose a search approach to the ONA planning problem, the other approaches mentioned above can also be used. Further research is necessary to find more about their applicability to ONA planning.

7.3 Planning for ONA

Recently, a number of ONA have appeared that conduct some sort of automated planning in two fields: customized routing and distribution of user data adaptations.

The system in [Choi00] chooses a route through the network that would improve the usage of link and node resources. The model consists of a network where each link and each node are associated with some cost and an intermediate computation that is to be performed somewhere in the network, given that not all nodes are able to perform this computation. The planning problem – to find the path with the smallest cost that contains at least one site where the computation can occur – has the same complexity as the graph shortest-path problem.

Conductor [Yarvis99A, Yarvis00] provides a central planning procedure that is run at the destination point of the connection using planning information on a fixed set of parameters for each link and node, user requirements specified in terms of link parameters and data characteristics, and the meta-descriptor and location of all available adapter modules. Although Conductor is able to plug in a variety of plan formulation algorithms, it currently employs a relatively cheap and simple planning algorithm that covers just simple cases, mostly because of an insufficient supply of well-developed planners.

One approach to automated planning was presented in CANS [Fu01]. The authors presented a method based on dynamic deployment of transcoding components (adapters), which takes as input only high-level specifications of component behavior and network routing characteristics. To our knowledge, CANS is the first attempt to build an automated planner for ONA. The complexity of the algorithm is a very important issue because of time limits on connection establishment. The CANS algorithm is based on a search in a stream-type graph with some simplification strategy allowing reduction of the graph. The complexity of the presented algorithm is claimed to be $O(p^3n^3)$, where p is the number of adapters, and n is the number of nodes. As we pointed out in Chapter 3, the complexity of our approach is $O(pn^2)$.