Conductor: A Framework for Distributed Adaptation❖

Mark Yarvis, Peter Reiher, and Gerald J. Popek¤
Computer Science Department, University of California, Los Angeles
{yarvis, reiher, popek}@fmg.cs.ucla.edu

Abstract

Most network applications provide poor service when the network's capabilities are below a minimum level assumed by the developer. As a wider array of network technologies become available, users will be increasingly frustrated with a lack of flexibility in such applications. The services provided by an application should be tunable to a level appropriate for the capabilities and associated costs of the underlying network. Other researchers have shown that proxy agents can tailor the communication pattern of an application to the characteristics of a network. Dynamic deployment of multiple adapting agents at various points in a network can extend these benefits. Conductor demonstrates an approach toward selecting an appropriate set of adaptive agents and a plan for their deployment. Conductor further allows arbitrary adaptations to be performed along the data path without reducing the reliability of the overall system.

1. Introduction

An increasingly large percentage of applications are network-based, providing a service to the user by communicating with one or more network servers. Application designers typically make assumptions about the minimum level of service required from the network, failing to consider mobile users and others who encounter varying degrees of connectivity.

Each link in a network may present a different level of bandwidth, latency, jitter, reliability, and security. These services are provided for some level of monetary cost and power consumption. If an application's use of the network is not curtailed when network characteristics are unfavorable, it may either (1) provide little or no useful service to the user, or (2) provide service at a cost greater than the worth to the user. Rather than trying to achieve the same level of service regardless of the properties of the network, an application's use of the network should be controlled, resulting in a degraded level of service.

Degraded service may require a compromise in timeliness, quality, completeness, or reliability of the user's data. However, the appropriate type and level of service degradation may vary. For instance, consider the distillation of a color map for quick download. A full-resolution black-and-white map would still allow the user to read the street names. On the other hand, if golf courses were marked in green, color may be more important than resolution. Selection of an appropriate compromise must be dynamic and controllable, depending on the user, the application, and the task at hand.

A variety of adaptations are possible to achieve a balance between cost and benefit for the user. Common techniques include compression, encryption, distillation (filtering), prioritization, pre-fetching, caching, or buffering. In addition, specific properties of protocols can be altered. For instance, a data transfer that normally uses small individually acknowledged messages could be augmented to include a sliding window or less frequent acknowledgement.

Advances in network technologies only increase the need for adaptation. Faster networks bring new, bandwidth-hungry applications, but fast wires won't reach everywhere. Mobile users, in particular, are always at a disadvantage as wireless networks continue to be slower, less secure, and more power-consumptive than wired networks. As the gap widens between the best and worst available connectivity, there is an increasing need for applications to adapt their services accordingly.

At the same time, networks are increasingly heterogeneous and complex. Techniques for adaptation have typically focused on the "last mile," assuming that a single host connects to a well-connected network over a single poor link [5, 6]. Successes in these systems have shown that gracefully degraded services are desirable. But, connectivity is becoming more complex with increasing instances of home computer networks, inter-building wireless links, and wireless workgroups. Success requires a more general solution for deploying adaptation.

Conductor is a piece of middleware that allows the operating system to provide adaptation as a service to applications. It includes a framework and a set of protocols for deploying adaptor modules into a network. Conductor is fully transparent to applications, allowing easy addition of

❖ This work was partially supported by the Defense Advanced Research Projects Agency under contract DABT63-94-C-0080.

¤ Gerald Popek is also associated with PLATINUM *technology, inc*.

new applications and new network technologies. It includes a distributed planning algorithm to determine what combination of adaptors is appropriate, and where they should be deployed. Finally, Conductor employs a unique reliability mechanism that allows a data stream to be arbitrarily adapted at multiple points, without compromising reliability.

2. Design Principles

The following design principles allow Conductor to apply adaptation in a wide variety of circumstances. While other adaptation systems adopt some of these principles, Conductor is unique in that it adheres to all of the principles.

• **Arbitrary Adaptations**: Adaptors may arbitrarily change the data stream.

In particular, an adaptor may add, delete, or modify portions of a data stream and may maintain state. This property makes it difficult to detect or to recover from an adaptor failure.

One proposed approach provides resilience to failures only for additive adaptations [7]. For example, forward error correction codes might be added when a link is particularly unreliable. Since this adaptation is additive in nature, adaptor failure only removes the possible benefit. Conductor's reliability mechanism removes this restriction, allowing any type of adaptor to be deployed.

• **Transparent**: Applications should not be required to be aware of or to control adaptation.

Adaptability could be built directly into applications, allowing applications to alter their behavior and interface as well as their use of the network according to the available connectivity. Such applications can provide adaptation-related control directly to the user. However, building adaptability directly into applications limits the list of tolerable network characteristics to those anticipated by the programmer.

Instead, Conductor provides application-transparent adaptation by operating on application-level protocols, external to applications. Upon final delivery to a client or server, the data stream must conform to the expected protocol. This is not to say, however, that all adaptations must be user-transparent. For example, the frame rate of a video stream could be cut in half prior to transmission, and later recovered via frame duplication, without violating the protocol expected by the application. Such an adaptation poses no new application requirements. However, since the quality of the user experience is affected, the user must have control over adaptation. Such control can be provided external to the application.

Application transparency allows adaptation to be applied to off-the-shelf applications and frees developers of custom applications from the complexities of adaptation. In addition, as new network technologies or user requirements develop, no application changes are required. In effect, transparency separates adaptation technology from application technology.

Although Conductor provides transparency, it can become less transparent by adding an optional API through which applications influence selection and operation of adaptations. Use of this API allows applications to more tightly integrate adaptation, allowing direct control of adaptation and a greater ability to represent different service levels to the user.

• **Composable**: Adaptors should be specialized, containing a single algorithm or function, allowing for adaptor composition.

The types of adaptations suggested in Section 1 are essentially orthogonal. By maintaining separation between encryption and compression algorithms, for example, we allow easy deployment of new algorithms in combination with existing adaptations and support for different combinations of security and compression levels that might be required.

In addition, it is important to separate protocol-specific adaptations (e.g., MPEG frame dropping) from protocol-independent adaptations (e.g., Lempel-Ziv compression). Otherwise, the number of adaptors required would be multiplicative in the number of protocols and the number of possible network conditions.

Because some adaptors may require a partial ordering with others, modularized adaptors require careful deployment. For instance, compression should normally occur before encryption.

• **Distributed**: Adaptation is most effective if it can be deployed into a network.

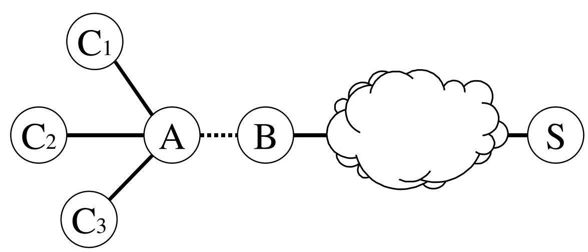

Figure 1: A sample network requiring distributed adaptation.

Consider the network in Figure 1. Clients C_1, C_2, and C_3 wish to perform database queries on server S. Assume an ISDN link connects nodes A and B. The clients could employ pre-fetching to minimize latency in responses from S. Since pre-fetching seeks to use spare bandwidth on the channel between nodes A and B to fulfill a predicted future need, it can only be effectively provided at node A. Node C_1 cannot determine when to pre-fetch because it cannot detect whether the ISDN link is free, or busy with an actual request from C_2. Moreover, a result from a pre-fetch may be of use to any of C_1, C_2, or C_3, so it should be cached at node A.

As another example, node A might be a cellular phone providing on-demand connectivity to nodes C_1, C_2, and C_3 via a personal-area network [1]. The clients wish to fetch Web pages and periodically request stock quotes over the cellular link. Since the cellular link is expensive, when the link is not already established for Web requests, the user may wish to provide stock quotes from a cache or queue these requests until several are pending. As before, only node A can perform the desired adaptation.

These examples demonstrate cases in which adaptations that might sometimes be performed on a client must instead occur on a node within the network. A server may not be an optimal choice either. Because a server may have many clients, it cannot be expected to have the resources to support a large number of adaptations. In addition, a server may not wish to accept arbitrary code (despite encapsulation) from any arbitrary host. Thus, clients and servers may wish to offload adaptation to more appropriate nodes in the network.

• **Multi-node**: In some cases, multiple nodes may be required for adaptation.

Consider again the network in Figure 1. Node A could prioritize client requests before transmission over the cellular link. Meanwhile, Web pages that are returned may contain very large images, requiring a compression adaptation at node B. Decompression could occur at either node A or at the requesting client. In either case, multiple nodes are required for this adaptation.

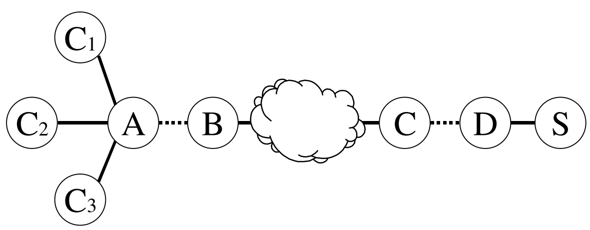

Figure 2: A sample network requiring dynamic multi-node adaptation.

At the same time, the locations at which adaptation is required cannot be fixed. Consider another network, shown in Figure 2, where the link from D to C also has poor bandwidth. In this case, compression would be best applied at node D rather than node B, avoiding multiple compression and decompression operations. Thus, a set of multiple, dynamically selected nodes may be required to provide the desired effect.

3. Conductor Design

Conductor allows a connection-oriented channel between a client and a server to be transparently adapted. Adaptation takes place by operating on the data and protocol in the stream. Application transparency is achieved by ensuring that the data stream visible at either endpoint conforms to the protocol expected by the application. That is not to say, however, that the stream received at one endpoint will be identical to the stream transmitted from the other endpoint. Adaptors are permitted to create, delete, modify, or delay any stream content, so long as the appropriate protocol is presented to the application. User-transparency is, therefore, not necessarily preserved and depends on the adaptations involved.

In addition, when a fundamental characteristic of a protocol is inappropriate for the network characteristics, an entirely new protocol can be created. Pairs of adaptors are employed to convert from the original protocol to the new protocol and back, providing transparent protocol conversion. Such conversions can be nested or combined sequentially in constrained ways.

Adaptors can be freely deployed on the endpoint nodes or on nodes in the network. Conductor is primarily a runtime environment, present on network nodes, that allows adaptive agents to be deployed along the data path. Toward this goal, Conductor provides facilities for interception of connections, node and network monitoring, adaptation planning, reliability, and support for deploying distributed adaptive agents.

Conductor allows adaptations to be performed on any Conductor-enabled node on the path from the client to the server. The Conductor framework can be present on client and server nodes, as well as other nodes within the network. Conductor assumes that nodes supporting adaptation are sprinkled throughout the network. Ideally, Conductor would be deployed onto special hardware within routers, which frequently hold the most complete information about network status. It is not necessary for all routers to be Conductor-enabled, nor would all router traffic be affected by Conductor. Nodes other than routers can also be Conductor-enabled. A Conductor-enabled node must be willing to participate in planning algorithms and can volunteer to adapt particular data streams. No minimum processing or storage requirements are assumed for Conductor nodes. Available resources are taken into account when determining where adaptors should be deployed. It will be advantageous for administrators to place additional computing and storage resources at key Conductor nodes.

Conductor-enabled nodes will include an observation component, using one of several existing technologies to monitor the availability of local node resources, such as spare CPU cycles, and the conditions and capabilities of adjacent network links [2]. For each Conductor-controlled connection, the data path will be divided into several sub-connections, one between each pair of nodes actively adapting the data stream. Adaptors are then deployed to each of those nodes.

3.1. Adaptors

To allow adaptor modules to be deployed dynamically, they are written in Java. Each Conductor-enabled node can store adaptor modules to be deployed along any data stream that passes through that node.

Each adaptor has an input protocol and an output protocol, and performs some function on a unidirectional data stream. Adaptors can be deployed in pairs, one adaptor converting from protocol A to protocol B, the other converting back to protocol A. Adaptor pairs can by loss-less, delivering unchanged data at the output, or lossy, delivering different data (but the same protocol). Adaptors that input and output the same protocol can also be deployed unpaired.

Each adaptor module includes self-describing meta-information which includes four components that insure proper deployment. The first two components indicate what protocol(s) the adaptor can operate on and the node resources required by the adaptor (as measured by the observation component).

The third component specifies the applicability of an adaptor to the network. For each link characteristic monitored by the observation component (e.g., bandwidth, security level, jitter), an input level and output function can be specified to describe the pre-conditions and post-conditions for deploying that adaptor. For instance, an encryptor might specify that it expects a low security level at its input and that it provides a higher security level at its output. For paired adaptors, the final output function is based on the initial input value. In this case, a decryptor would return the security level to the level measured at the input of the paired encryptor.

The final meta-descriptor component specifies a set of interoperability parameters, which restrict adaptor combination. As before, these parameters specify an expected input level and an output function, indicating pre and post conditions. Two types of interoperability parameters are allowed: standard and extended. Standard parameters describe protocol-independent characteristics of a data stream and must be specified by all adaptors. For instance, a compression adaptor might require a certain amount of compressibility to be deployed and would result in a less compressible output. An encryption adaptor would have no compressibility requirement but would create a nearly incompressible output. Such a specification restricts compression to occur before encryption. Extended parameters are analogous, but are optional and express protocol-dependent data characteristics (e.g., frame rate). For most adaptors, composition requires that each adaptor specifies the acceptable input levels for all extended parameters produced by its neighbor. Protocol-independent adaptors are an exception to this rule.

3.2 Planning

When a new client-server connection forms, Conductor will provide a cost-benefit compromise for the user by selecting and deploying the most appropriate set of adaptors. Conductor-enabled nodes along the network path between the client and server form the *planning path*. These nodes are involved in planning activities.

Planning could be accomplished by allowing each Conductor-enabled node to make a locally optimal plan, deploying adaptors to alleviate deficiencies in local links. This approach results in low planning latency, but can produce an adaptation plan that is not globally optimal. Reconsider the network in Figure 2. If the two dashed lines represent low bandwidth links, local planning would result in two pairs of compression and decompression adaptors, rather than a single pair spanning both links. Further gossiping between nodes could identify the redundancy, and the four deployed adaptors could be replaced by two. However, since adaptor deployment (and removal) is a heavyweight operation, it is preferable to pay a higher up-front cost for a globally optimized plan.

Because adaptors can cache data, optimistically generate data, or buffer data, some level of service may be provided despite a partition between a client and server. Therefore, we must not require global planning. Instead, Conductor employs per-partition planning. Within each network partition, a partition-wide plan is generated.

For partition-wide planning, all relevant information is gathered at a single node, a plan is generated and distributed to all nodes, adaptors are deployed, and data begins to flow through the adapting nodes. If network conditions deviate from the input specification of any deployed adaptor, replanning occurs, possibly resulting in replacement or addition of adaptor modules.

Although information gathered during planning can be potentially large, it is primarily connection-independent and somewhat static. In addition, since network routing tends to follow regular patterns, planning paths will tend to overlap. By caching the planning information received from each node in the path, the need for transmission, and therefore the planning overhead, can be reduced.

The information gathered for planning includes a fixed set of parameters for each link and node, user requirements specified in terms of link parameters and data characteristics (as described by adaptor interoperability parameters), and the meta-descriptor and location of all available adaptor modules. Notice that the node resource requirements contained in each adaptor's meta-descriptor constrains the nodes on which it can be placed. Further, an adaptor's interoperability parameters constrain the combinations of adaptors that are permitted. Finally, the user requirements determine which adaptors would be desirable and needed.

Because all planning information is gathered at a single node, a variety of plan formulation algorithms could be plugged into the system. We could consider a greedy algorithm that, with each iteration, considers the largest mismatch between link characteristics and user requirements. The algorithm would then select an adaptor (or pair) which alters the link characteristic to most closely match the user's requirements. As the algorithm proceeds, it is constrained in new selections by the interoperability parameters of previously selected adaptors. Clearly, such an algorithm would produce non-optimal plans. An alternative algorithm could build a tree of all such possible selections and evaluate all leaf nodes (plans) based on adaptor deployment costs, adaptation overhead, and the degree to which each plan matches the user's requirements. This algorithm could be expensive. Currently, Conductor employs a simple and cheap planning algorithm, and we are exploring other algorithms.

3.3 Reliability

A reliable data stream provides exactly-once delivery semantics. Since adaptors can arbitrarily change data in transit, data delivered may be arbitrarily different from data transmitted (at any two points along the connection path). It is difficult to define exactly-once semantics in the presence of adaptation. If an adaptor, proxy node, or link were to fail, a comparison between data received downstream and data sent from upstream will not necessarily indicate an appropriate point of retransmission

Moreover, since adaptors may be paired and maintain state, loss of one adaptor module may require the addition or deletion of other adaptor modules to restore symmetry and correct composition. Without further support, adaptor loss could lead to data loss. Conductor employs two mechanisms to allow proper failure recovery: *semantic segmentation* and *protocol hierarchies*.

Like many systems, Conductor uses segments as a unit of retransmission. Typically, segments are necessarily immutable in transit. Conductor extends this model, allowing in-transit adaptation. A segment in Conductor contains a semantically meaningful portion of data. For example, in a video stream, a segment might contain a single frame. An adaptor may choose to turn a color frame into a black-and-white frame by modifying data in a segment while preserving its semantic meaning. Since a segment is the unit of retransmission, each frame will be received exactly once, in one format or the other.

Segmentation occurs dynamically, according to the properties of the data and the adaptations. Initially, a data stream is logically composed of single-byte segments. To preserve the semantic retransmission property, an adaptor is required to contain each data stream modification within a given segment. If a desired modification would cross segment boundaries, the adaptor first combines the adjacent segments into a single segment. The new segment should represent all of the semantic meaning contained in the original segments. The new segment is labeled in a way that indicates the composition that occurred. Failure recovery can be accomplished at the point of failure by canceling any partially transmitted segment and requesting retransmission to begin with that segment. Any version of a segment or the composed segments can be used for retransmission.

Although data loss is prevented using segmentation, Conductor must also preserve adaptor pairing. Consider the adaptor pairs in Figure 3. Adaptors BC and CB are paired, converting protocol B to a new protocol C. If BC fails, either it must be replaced, or CB must be removed. Since some internal state may be lost with a failed adaptor, BC cannot, in general, be replaced. Instead, CB is removed. In addition, after BC fails, any data approaching from CB cannot be converted back to protocol B, so retransmission occurs.

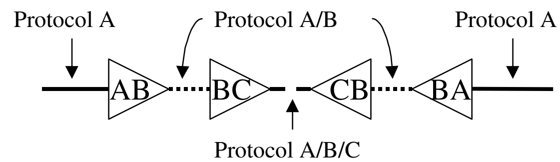

Figure 3: Protocol hierarchies.

When a failure occurs, Conductor must be able to determine which adaptors to remove. Since planning activities can be localized by network partitions, a given node may not have global knowledge of deployed adaptors. Instead, Conductor associates a *protocol hierarchy* with each link (see Figure 3), indicating the hierarchy of adaptations performed to achieve the current protocol. When a link, node, or adaptor failure occurs, the data path is first spliced back together. The new protocol hierarchy for the link is determined by the greatest common ancestry between the hierarchies upstream and downstream of the failure. Adaptors on either side of the failure that do not conform to this new protocol hierarchy are removed. Notice that when adaptors are composed (as in Figure 3), failure of an outer adaptor will require inner adaptors to be removed as well, to preserve proper composition.

4. Development Status

Conductor's primary mechanisms and algorithms are fully designed. Development of the Conductor runtime environment for Linux is underway.

Currently, Conductor is able to intercept TCP connections, form a planning path among Conductor-enabled nodes, perform simplified plan formulation, deploy simple adaptors along the data path, and adapt the data stream. We are expecting to complete a prototype, including support for more sophisticated plan formulation and reliability, in 1999.

5. Related Work

A variety of adaptive techniques have already been explored in various systems. These systems can be loosely divided into three categories.

• **Application-oriented**. Some systems provide a set of programming tools to aid applications in adapting to changing network conditions. Odyssey [3] provides feedback to the application about network conditions and allows the application to request different levels of data fidelity. The Rover Toolkit [4] allows applications to be subdivided, placing some functionality on either side of a low-quality link. By allowing applications to be smarter about network concerns, these systems allow user-controlled application-specific tradeoffs to be made about network use. However, these systems require programmers to consider the variety of environments the application is likely to encounter and the types of service compromises users will accept.

• **Proxy-oriented**. The Daedalus Project has had great success using their Transformation, Aggregation, Caching and Customization (TACC) architecture to construct adaptive proxies [5]. Their TranScend proxy and the Top Gun Wingman system have demonstrated, in real use, that adaptation can provide users with a cost-to-quality compromise. TACC allows proxy services, composed of worker modules, to execute on a compute cluster. Columbia University researchers have constructed a general-purpose proxy framework that allows dynamic loading of adaptor modules to a proxy node, giving even greater control to users [6]. Both of these systems focus on "last mile" issues, adapting data at a well-connected proxy node before transmission across a low-quality link. As a result, both systems are currently limited to a single proxy service per connection and make no provisions for distribution of adaptations or resistance to proxy failure.

• **Protocol-oriented**. Several researchers have focused on adapting protocols to make adaptation a transparent network function. Protocol Boosters [7] allow multiple booster objects to be composed together and dynamically distributed along a data path, augmenting the transmission protocol. Resilience to adaptor failure in this system is only provided in the case of additive operations, which leave the initial data stream intact. Boosters of this type are limited in the types of adaptations they can provide. In a different approach, Transformer Tunnels [8] from the Dataman project provide per-link adaptation by allowing pairs of nodes to create adaptive tunnels. All data flowing through a tunnel is similarly adapted. Tunnels can, therefore, be administratively deployed wherever low-quality network elements exist. This approach, while extremely efficient, provides very little flexibility and responsiveness to the requirements of individual users.

Conductor has benefited from the lessons of these systems and combines many of their best properties.

Emerging research in *active networking* [9] may provide a future direction for Conductor. Active networks allow programs to be attached to individual packets or to be deployed into network switches. Although active network code is frequently lighter in weight than the typical adaptation envisioned for Conductor, techniques developed for Conductor may prove useful for active networks.

6. Conclusions

In the future, networks will require flexible adaptation technology, allowing applications to gracefully degrade their services. Conductor allows an operating system to provide protocol adaptation as a transparent service. Conductor provides a general mechanism to select and dynamically deploy combinations of adaptive agents to multiple points in a network. In addition, Conductor maintains resilience to failure while allowing arbitrary adaptations. These properties are crucial to providing the exact cost-benefit balance desired by each user in a complex network.

7. References

[1] Jaap Haartsen, Mahamoud Naghshineh, Joh Inouye, Oalf J. Joeressen, and Warren Allen, "Bluetooth: Vision, Goals, and Architecture," *Mobile Computing and Communications Review*, Oct. 1998, 2(4):38-45.

[2] Pradeep Sudame and B.R. Badrinath, "On Providing Support for Protocol Adaptation in Mobile Wireless Networks," Rutgers University, Department of Computer Science Technical Report, DCS-TR-333, July 1997.

[3] B. Noble and M. Satyanarayanan, "Agile Application-Aware Adaptation for Mobility," *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, St. Malo, France, Oct. 1997.

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

[5] Armando Fox, Steven D. Gribble, Yatin Chawathe, and Eric Brewer, "Adapting to Network and Client Variations Using Infrastructrual Proxies: Lessons and Perspectives," *IEEE Personal Communications*, Sept. 1998, 5(4):10-19.

[6] Bruce Zenel and Dan Duchamp, "A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment," *Proceedings of the Thrid Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM* '97), Oct. 1997, pp. 248-259.

[7] D.C. Feldmeier, A.J. McAuley, J.M. Smith, D. Bakin, W.S. Marcus, T. Raleigh, "Protocol Boosters," *IEEE JSAC, Special Issue on Protocol Architectures for the 21st Centruy*, Apr. 1998, 16(3):437-444.

[8] Pradeep Sudame and B.R. Badrinath, "Transformer Tunnels: A Framework for Providing Route Specific Adaptations," *Proceedings of the 1998 USENIX Annual Technical Conference*, New Orleans, Louisiana, June 1998.

[9] 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*, Jan. 1997, 35(1):80-86.