Infocom '99

The University of Kansas

Information and Telecommunication Technology Center

Technical Report

Active Networking Services for Wired/Wireless Networks

Amit Kulkarni
Gary Minden

ITTC-FY98-TR-12161-02

November 1997

Active Networking Services for Wired/Wireless Networks[1]

Amit B. Kulkarni and Gary J. Minden
Department of Electrical Engineering and Computer Science
University of Kansas, Lawrence, KS 66045

Abstract - An active networking architecture provides the infrastructure for applications to inject user programs into the nodes of the network. This enables customization of the network nodes so that application-specific services can be downloaded into the network in the form of new protocols. In this paper, we look at the different kinds of services that applications require from the network. We study these services to identify common characteristics so that services can be grouped into classes of protocols. We argue that this study permits us to develop a methodology that helps in rapidly designing and deploying new protocols.

I. INTRODUCTION

Traditional networks have the drawback that the intermediate nodes are closed systems whose functions are rigidly built into the embedded software. Therefore, development and deployment of new protocols in such networks requires a long standardization process. The range of services provided by the network is also limited because the network cannot anticipate and provision for all needs of all possible applications. The inflexibility of the current network architecture severely inhibits protocol innovation.

Active networking [1][2] offers a different paradigm that enables programming intermediate nodes in the network. In an earlier paper [3], we defined a network to be active if it allows applications to inject customized programs into the network to modify the behavior of the network nodes. This allows applications to customize the network processing and adapt it to the application's immediate requirements. This enables new protocols and new services to be introduced into the network without the need for network-wide standardization. To understand the advantages and drawbacks of an active networking environment, we constructed a prototype active network implementation [4] that serves as a vehicle for deploying new protocols and study the performance of the components of the prototype. In this paper, we address the issue of the actual development of new protocols for the active network. The goal is to develop a methodology for introducing new protocols in an active network and study the performance of the sample protocols.

The remainder of the paper is organized as follows. The next section describes the problem motivation. Section 3 briefly touches on the concept of active networking and the essential components of an active network. Section 4 develops the concept of protocol classes. Section 5 describes some of the problems of wireless networks and suggests solutions in the form of new protocols based on the protocol classes defined in the previous section. Some experimental scenarios and performance of the new protocols are shown in Section 6. The paper concludes with an overview of related research and a summary.

II. BACKGROUND AND MOTIVATION

In recent times, substantial research has focused on performing application related processing inside the network. Research on booster protocols [5] attempts to selectively add protocol stubs or "boosters" in the network on a per-application basis. Boosters are protocol modules that implement a proprietary protocol and are statically configured into the protocol stacks of the network nodes or the end systems. The protocol modules can be combined with other protocol modules dynamically to provide on-the-fly customization of the network. Another example of application-specific processing inside the network is the research on active or adaptive caching strategies being developed by [6]. Client requests are sent to a cache that actively decides whether to respond to the request directly or pass the request along to other caches or the origin server based on its knowledge of the network state.

New standards developed for video send MPEG coded video in layered form. By prioritizing the layers, it is possible to maintain real-time connectivity in times of network congestion by dropping packets containing the lower priority layers. Similar strategies are being used for audio transport wherein the signal components are separated based on their level of contribution to the original sound. Signal components that do not contribute heavily are placed in lower priority packets that are specially marked for discard if congestion occurs. In the Wireless ATM Voice/Data project [7] at the University of Kansas, two or more voice packets are compounded and sent as one voice packet reducing the bandwidth by the factor of the compounding. Thus we see that in all the examples, the application tells the network in a limited way how to handle its data.

Other research focuses on the advantages of putting processing in the network that was traditionally performed at the end hosts. In the Distributed Sensor Data Mixing project

[1] This research is partially funded by the Defense Advanced Research Agency (DARPA) under contract F19628-95-C-0215.

[8] at Lancaster University, remote sensors such as microphones and antennas, collect data and transmit them to receivers on the network. Instead of having each receiver perform its own mixing of the transmitted data, some of the mixing is done within the network on the input signals that pass through the network node at approximately the same time. If the mixed signal is smaller than the sum of its constituents, then it reduces bandwidth requirements and the processing to be done at the receiver. These efforts attempt to create new network protocols to handle application-specific processing. However, in traditional networks, design and deployment of new protocol standards take time. Active networking provides a platform for introducing and testing new protocols into the network without a long standardization process. Active networking prototypes developed at MIT [9], University of Kansas [4] and Georgia Institute of Technology [22] have actually implemented some of the ideas mentioned above as new protocols in their prototypes. This experimentation suggests that protocols are nothing but services provided by the nodes of the active network. In an active network, applications have the ability to access these services and customize them for their needs.

Our goal in this paper is to identify and classify protocols based on functionality that a network generally needs to provide to applications. Such taxonomy enables us to create *protocol classes* for services that an application is apt to demand from the network. Essentially, these classes describe properties of the protocols that implement the services that applications need. Currently, the taxonomy contains protocols and classes that are generally prevalent in current networks. Even so this study is important because a thorough study of these classes will eventually lead to a better understanding of the manner in which protocols must be designed and deployed in active networks.

III. Active Networking Concepts

Traditional networking implementations follow a layered model that provides a well-defined protocol stack. Many implementations, for performance purposes, embed the networking implementation across network interfaces, the operating system, runtime system, and/or the application. Most implementations provide a fixed protocol stack that is determined by a long standardization process often taking many years and is fixed when the final system is constructed. The time delay from the conceptualization of a protocol to its actual deployment in the network is usually an extraordinarily long process. Active networking offers a technology where the application can not only determine the protocol functions as necessary at the endpoints, but one in which applications can inject new protocols into the network for the network to execute on behalf of the application. The protocols presented in this paper are developed on the prototype implementation of an active network architecture built here at the University of Kansas. The details of the architecture are described in [4]. In this paper, we cover only those aspects of the architecture as are necessary for the understanding the concepts.

A. SmartPackets

In an active network, data packets are information entities. These entities, called ***SmartPackets***, contain a destination address, user data, and methods that are executed locally at any node in the active network. Since architectures like the x-kernel [10] and Horus [11] allow us to compose complex protocol stacks from single-function protocols, we can think of SmartPackets as carrying customized protocols that have to be fitted in with protocol modules at the network nodes. The code in the SmartPacket can be in any executable format and it is executed at the node if the node has the correct processing environment. In our implementation, the code of the SmartPacket is a Java [12] class that describes the methods that act on the user data.

B. Active Nodes

Nodes in an active network are called **active nodes** because they are programmable elements that allow applications to execute user-defined programs at the nodes. Active nodes perform the functions of receiving, scheduling, executing, monitoring and forwarding SmartPackets. When a SmartPacket arrives at an active node, it is scheduled for execution. A separate environment is required for each invocation to prevent undesirable interactions and malicious access to node resources. Active nodes export a set of resources and primitives that can be used by user programs. Active nodes also enforce constraints on the actions that can be performed by user code. Thus an active node enforces a time limit on the execution of SmartPackets to prevent runaway user programs and on the total memory that can be requested by a SmartPacket. User code has restricted access to certain internal information such as the routing tables, buffer space information and available link bandwidth on the node's interfaces. Users can utilize this information to develop application specific strategies to combat congestion or implement a custom routing policy. SmartPackets are also allowed to leave behind information in the form of small state for trailing SmartPackets to utilize.

IV. Protocol Classes in Active Networks

The implementation the prototype active network has enabled us to develop and test new protocols. We have implemented new protocols for routing (a beacon routing protocol based on the geographical routing concept [13]), caching (smart HTTP) and mail transfer (smart SMTP). We have also developed [4] active versions of ping and traceroute that are efficient as well as intelligent. We have closely followed the experiments conducted on prototype active networking platforms at MIT [9] and Georgia Tech [22]. The experimental protocols described in the above studies were developed specifically for a chosen application. But a closer study of these protocols indicates that while each protocol seems application-specific, it generally possesses characteristics that are common to some other protocols. All such protocols can collectively form a class. Classification enables us to isolate properties shared by members of a class. This facilitates rapid design of new protocols and their seamless introduction into the network.

Consider the example of a new protocol being developed that gets identified as belonging to one of several pre-defined classes. If all active nodes advertise the protocol classes that they support, then we are assured that the new protocol will run on the active nodes that support the protocol class. We can quickly build a prototype of the protocol using the interfaces and the node primitives defined for the class and start experimentation. The development and deployment times for the new protocol are thus vastly reduced. The experimental results and performance measurements obtained from the prototype are then used to fine-tune the protocol. Grouping protocols into classes allows us to develop a methodology for designing new protocols and for using existing protocols. At this stage, we have identified the following key issues to consider while developing a methodology for each class: (1) architecture for deployment, (2) common interfaces, and (3) node primitives required. The architecture refers to the placement (location) of the protocol services in the nodes of the active network. The structure of the SmartPacket and its design make up the interface while the node primitives describe the requirements of the protocol before it executes at an active node. It is important to find out if there is a topology for the distribution of a protocol inside the active network that is the most efficient. For example, protocols designed to match data rates on either side of a wireless gateway have to be deployed at the wireless gateway itself. During the design phase, one might discover that the protocol has to be deployed at select nodes that possess certain characteristics. If so, is there a way by which such nodes can be dynamically identified?

The grouping of protocols into classes helps in identifying common interfaces. We can then define a structure for the SmartPackets that implement protocols belonging to the particular class. Clearly, this enables rapid prototyping. Protocols of the same class generally require the same set of services from an active node. For example, protocols belonging to the bridging class require buffer space because the protocols combine packets and sometimes they have to hold a packet from one of the streams while packets from the other stream(s) arrive. It is therefore useful to identify the common services required for the class and the primitives that an active node must provide for SmartPackets to avail of those services. We have identified the following classes of protocols: **filtering, combining, transcoding,** security **management, network management, routing control** and **supplementary services**. We discuss their features in detail and outline a methodology to develop protocols belonging to each of the classes.

A. *Filtering Class*

The **filtering** protocol class encompasses all those protocols that perform packet dropping or employ some other kind of bandwidth reduction technique on an independent per-packet basis. We see that compression protocols and the transmission of layered MPEG falls in this category. Protocols belonging to the filtering class are primarily developed to reduce bandwidth requirements of the application data. Bandwidth reduction techniques are always required whenever there is a severe rate mismatch. This typically occurs at interfaces where there is an order of magnitude difference in the speeds on opposite sides of the interface e.g. the interface between wired and wireless networks. In such cases, it is obvious that the protocols have to be deployed at the interface gateway and therefore it must be included in the topology.

Applications identify nodes for deploying the filtering code by using SmartPackets that "sniff out" the nodes within a set of nodes at which a rate mismatch occurs. Alternately, the connection path is primed by installing the protocol at all nodes on the path to tackle any congestion problems. Since protocols of this class are designed primarily to reduce bandwidth requirements, the active node must supply primitives: (1) to find out available bandwidth over a given interface; (2) to find the bandwidth capacity of all interfaces; and (3) management of small state involves providing primitives for the creation of, the storage of information to and the retrieval of information from the small state.

B. *Combining Class*

The class of **combining** protocols has the property of combining packets that may come from the same stream or from different streams. The Wireless ATM Voice/Data project [7] combines two or more packets from the same stream to form a single packet that is forwarded to the next hop. This is not a member of the filtering class because packets from the same stream are being combined. The Sensor Data Mixing project [8] is another example because it combines different streams into one. Research on caching techniques also falls in this category because caching combines common requests from separate streams as one consolidated request and then multicasts the reply back to the requesting parties. Protocols belonging to the bridging class are best deployed close to the sources of the data streams to enable early mixing of the data streams since the mixing process generally reduces the bandwidth of the resultant data stream. This process is the inverse of multicasting and some of the architectural issues of multicasting seem to be relevant here. Combining is computation-intensive and therefore active nodes deploying such a protocol must have sufficient processing power. They must also have sufficient memory storage because combining sometimes involves storing packets from one stream until packets from the other arrive at the node. Therefore the interfaces that active nodes have to provide for this class of protocols are (1) to find memory available for the SmartPacket; (2) management of small state; and (3) cloning/duplication of SmartPackets for multicasting.

C. Transcoding Class

Protocols that transform user data into another form within the network belong to the class of **transcoding** protocols. Examples of such protocols include encryption protocols and image conversion protocols. These protocols are CPU-intensive and therefore require nodes with sufficient computing resources. Encryption protocols are generally deployed only at the end-points of a connection As mentioned before, protocols of this class are primarily processing functions and therefore the primitives desired are (1) available memory; and (2) computing resources.

D. Security Management Class

The ability of the user code to access node resources raises questions about security. Active nodes must prevent SmartPackets from over-consumption of resources such as CPU time and memory storage. Active nodes must prevent SmartPackets of one application from interfering with the execution of other unrelated SmartPackets. SmartPackets must also be prevented from unauthorized access to node resources such as internal tables. But management packets i.e. SmartPackets that manage services at the active node, require access to internal data structures. Therefore, active nodes implement levels of security to authorize and monitor accesses. A security association is established by a security protocol to determine the level of security at which a SmartPacket executes. Security protocols use security infrastructures such as key exchange or identity certificates to create and authenticate a security association. The association also determines the authorization that SmartPackets belonging to the association have to the active node's resources. The authorization includes or excludes certain rights of access such as read/write/create permissions to internal tables or memory storage, or ability to create or define new properties for the association and/or modify existing properties. Therefore, the following primitives are required by protocols belonging to this class: (1) to create, maintain, and destroy security associations; and (2) to establish, extract, grant and revoke authorizations and privileges to SmartPackets.

E. Network Management Class

The programmability of the nodes in an active network enables the creation of self-configuring, self-diagnosing and self-healing networks. This involves actions such as alarm and event reporting, accounting, configuration management, and workload monitoring. The advantage of using active network to perform such functions is that it is possible to capture a consistent state of a node by sending a single SmartPacket that gathers all relevant information at one time. In SNMP, the management station sends successive SNMP requests to a node and the information gathered is later assimilated. The information gathered in this manner may not be consistent because there is an inherent latency involved in sending requests and getting replies. State at a node may change by the time the next request arrives and therefore the replies cannot necessarily be correlated to reflect the correct state of the node. Some state information could be such that it is required to be gathered only if some other state variable possesses a certain value. The logic required to perform such an action must has to be previously coded at the node or the management station is forced to make successive requests, once for the independent state variable and then for the dependent state variables.

Protocols performing such actions need a completely different set of privileges and architecture. The security association described earlier provides the privileges for the protocols to gather state, update node properties such as usage limits, and report state back to monitoring entities. The architecture involved in the deployment of these protocols requires dynamically establishing monitoring and monitored entities in the network. Alarm and event reporting functions have to be defined and installed at various nodes in the network. Accounting and performance information is gathered, stored and, in some cases, processed and reported to other entities on the network. The active nodes must provide management SmartPackets with the following interfaces: (1) to create, access and modify the state of the active node; (2) to establish events and state information to be gathered; and (3) to establish frequency and format of reporting event information. For example, event information can be formatted in Universal Logger Message [14] format.

F. Routing Class

Active networks provide applications with the benefit of overlaying several virtual topologies on top of the same physical network. Applications can use their own routing protocol that is derived from the virtual topology defined by the application. Thus we can implement several novel routing strategies such as geographical routing and information-based routing. This class of protocols is different from the network management class because in an active network, routing is an application service. Using custom routing strategies, applications create virtual network topologies that are overlaid on the physical network. Such virtual topologies are important for, say, mobile users to simplify handoffs and location management. The primitives that protocols of the routing class require are (1) information about the ports to neighboring nodes; (2) interfaces to receive port status messages; and (3) information about queue sizes at the ports.

G. Supplementary Services Class

Some protocols have the property of adding new features to the packets based on their contents. These protocols can be grouped into the **supplementary services** class. We use the term supplementary because these protocols do not alter the contents of the packet and usually add a new feature to the packet. An example of this protocol is content-based buffering which is an extension to the concept developed by the DataMan project [15][16]. In a wireless network, if connectivity to the mobile host is lost, incoming packets are buffered or discarded depending on the contents of the packet. Buffered packets are transmitted once connectivity is re-established. Content-based buffering increases efficiency of the protocol by discarding packets that contain real-time data that cannot be delivered in time. The typical requirements of these protocols are (1) finding available memory; and (2) management of small state.

V. Problems in Wireless Networks

The principal concerns of wireless networks are that the mobile hosts are typically resource-poor and have low-bandwidth, unreliable connectivity to the static elements. The rate at which data is transmitted to the client has to be limited to match the bandwidth of the wireless link to prevent queue overflows and potential data loss. Rate-limiting is achieved either by compressing the data before sending it over the link or by filtering out data packets. Applications alone can determine the most suitable method. Active networking addresses this problem by allowing the applications to install the required filter at the wireless gateway.

Wireless networks are considered unreliable because the links have changing bit-error rate (BER) and are prone to sporadic connectivity breakdowns. Forward error correction (FEC) is typically used to counter high BER. Adding FEC padding increases the size of the data packet being transmitted. Higher the BER, more the number of FEC bits required. But in the case of a changing BER, a constant FEC padding is not suitable. If the current BER of the link is low, then the extra FEC bits are an overhead. On the other hand, if the BER is too high, the FEC used is not good enough. Intermittent connectivity poses a different kind of problem. Incoming data packets have to be cached at the base station during times of disconnection and the unit must be re-integrated upon reconnection and the data packets transmitted at that time. The base station or gateway must possess a large buffer space to hold all data packets that arrive during the disconnection period.

A. Active Networking Solutions

We now attempt to derive solutions to the above problems by using the protocol classes defined earlier. Using a protocol belonging to the supplementary services class such as an adaptive FEC protocol solves the problem of changing BER. The BER of the wireless link is monitored continuously. Based on the current BER of the link, an appropriate number of FEC bits are added on the packet before transmitting it across the link. Thus the strength of the FEC padding depends on the BER of the link.

A protocol such as content based buffering which is a member of the supplementary services class can be used to counter intermittent connectivity. In content-based buffering, applications use SmartPackets that encode information about the time-sensitivity of the data (e.g. audio) and the maximum allowable delay that it can incur before it reaches its destination. If the mobile node is temporarily disconnected, the SmartPackets are cached at the base station by invoking a "sleep" method. For text data packets, the "sleep" method does nothing, but for timing-sensitive packets the encoded algorithm periodically checks the current time against the allowable delay and discards the packet if the delay time is exceeded. The bandwidth problem is tackled by the protocols of the filtering class. If the data is in packetized form like packet audio or layered MPEG, we use a packet dropping technique. For text and graphics data, we can use a adaptive compression protocol that varies the ratio of compression based on the available bandwidth.

B. Application-specific Filtering

We have utilized the filtering protocol class described above to create a new filtering protocol for applications in the MAGIC-II project (http://www.magic.net). We implemented the new protocol in the active network to make applications like Ibrowse and Terravision more responsive to network characteristics, especially for clients that are connected over low bandwidth links. These applications are image display clients that make requests for image data from remote servers to view imagery of terrain data stored on the servers. These requests are made as part of a frame. The requests in each frame correspond to tiles that are part of the current view. The image data is returned as a set of tiles that the client application processes to render the terrain. When the user is panning the terrain, the client application makes requests for a large number of tiles. If the user is panning very quickly, it is entirely possible that by the time some of the tiles reach the client, the user has no need for them because he has already requested a different set of tiles. In this case, the received tiles are dropped by the application. This is especially true when the client is running on a wireless host i.e. a host connected to the network over a wireless link. If the user is panning very rapidly, then the wireless gateway can become congested resulting in deteriorated performance. The network attempts to deliver all requested tiles to the client, including those that will eventually be dropped. An application-specific solution to this problem is to prevent tiles that are candidates to be dropped by the client, from being transmitted across the wireless link. This reduces the demand on network resources and enhances the ability of the application to respond to user input because there are more network resources available when it needs them.

The active network solution is to have the client customize the network by dropping filtering code at the gateway (Fig. 1) that maintains the identifiers of tiles that are currently requested. Tiles that are not part of the current request list are dropped at the gateway itself. This prevents congestion at the wireless gateway. Because only tiles from the current request list are allowed to pass through, the user is not limited by the capacity of the wireless link. When the user stops panning, the request list at the gateway corresponds to tiles in the

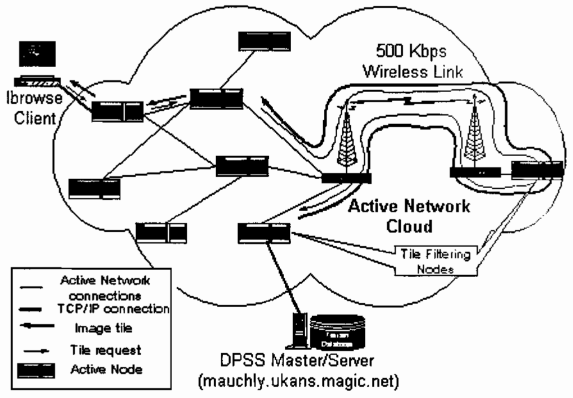


Fig. 1. Example of filtering class protocol

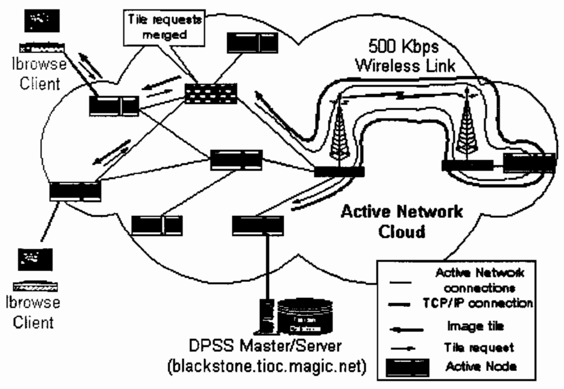


Fig. 2. Example of combining class protocol

current view enabling to pass through. The application is thus able to quickly synchronize up with the user's demand.

C. Application-specific Bandwidth Sharing

We have also created a new protocol called Request Merging protocol derived from the combining class of protocols. We trap requests from clients and transparently redirect them to nodes on which a proxy exists that keeps track of the requests made to the server(s) (Fig. 2). The proxy acts like a beacon and sends information about itself to neighboring nodes periodically as a special SmartPacket. The information includes location of the proxy, time sent, expiration time and number of hops to the proxy. When a request reaches an active node, it is redirected to the nearest proxy based on the information stored at the node. At the proxy, if an incoming request has already been made on behalf of another client, the current request is cached. The received reply is multicast to all clients.

The architecture is distributed, dynamic and survivable. It is distributed because there can be several proxies and the clients do not have to know of them beforehand i.e. they are "discovered" as the request is forwarded through the network. It is dynamic because the proxies are mobile. The architecture is survivable because if a proxy dies or the node on which the proxy resides goes down, the requests automatically propagate towards the destination until they encounter another proxy. The architecture also allows for cascading proxies so that a request made by an upstream proxy serves as an input request to a proxy downstream. The reply is multicast back in the same manner.

VI. Experiments and Performance Analysis

We conducted experiments to measure the performance benefits obtained by implementing the solutions described for the above scenarios. The topology of the network is as shown in Fig. 1 and Fig. 2. The network consists of 8 active nodes, a workstation running the Ibrowse application and another running the image server connected by 10 Mbps Ethernet. One of the active nodes is connected to the network over a 500 Kbps radio link. In all experiments, Ibrowse fetched 49332-byte image tiles from a remote server over the wireless link. A static route ensures that IP packets from the client to the server are always forwarded over the wireless link. In the active network, routing was controlled so that the route between the client and the server included the wireless link.

In active networking, we trade processing time at the network nodes for benefits gained by sending information rather than data through the network, which can be in the form of reduced bandwidth demand, better utilization of node resources such as buffers and queues and/or application-specific customization. Performance of the network is traditionally measured in terms of throughput and delay. But simply calculating performance in terms of tiles delivered per second is inappropriate. Because Ibrowse drops tiles that are outside the user's frame, a better measure is to find out the number of tiles that the application actually displays. Therefore we define ***effective throughput*** to be the rate at which information (i.e. data utilized by the application) is delivered to the application. This differs from the traditional interpretation of network throughput, which is the rate at which the network delivers data irrespective of its utility to the application at that time. In our prototype active network, all processing is performed at the application level. This adds overheads to the delivery of packets (tiles) from the server to the client. The result is that our prototype is unable to keep up with the standard network protocols, which are optimized and run in the kernel. Our throughput experiment (Fig. 3) verifies

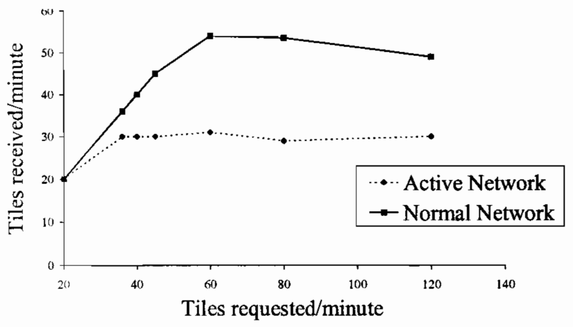


Fig. 3. Throughput performance comparison between traditional and active networks

that the baseline performance of our prototype is 1.7 times slower than the standard network in terms of throughput. The peak throughput obtained by the prototype is a little above 31 tiles per second whereas the throughput over the standard network peaks at 54 tiles per second. At request rates beyond 60, the capacity of the wireless link is a factor. The link capacity prevents the tiles from reaching the client as fast as the demand resulting in a backlog at the wireless gateway. From a perception standpoint, the user is unable to pan freely over the terrain. The application's response deteriorates until the user stops and allows all the bottlenecked tiles at the gateway to be transmitted over the link.

We do not directly compare the performance gain obtained by filtering vis-à-vis the standard network, because active networking is still in its infancy and even with the performance gain achieved through filtering will not be able to match the performance of the standard network. But since we are interested in quantifying the benefits of active filtering, we normalize the performance by running the experiment only on the active network implementation with and without filtering. We then use this metric to extrapolate on the gain that can be achieved when the performance of active networks approaches that of traditional networking protocols. We instrumented Ibrowse to report the number of tiles that arrive and the number of tiles that are dropped. The difference between the two numbers gives the "effective throughput" i.e. the number of tiles that are useful to the application. Fig. 4 shows the raw performance gain in tiles per minute. The improvement in effective throughput is marginal up to a request rate of 36 requests/min. As the request rate increases the filtering code allows only useful tiles to be transmitted over the link. This frees up bandwidth over the wireless link for later requests and therefore increases effective throughput. But at a rate of 60 requests/min, the filtering code prevents almost all tiles from reaching the client resulting in marginal effective throughput.

The performance of the request merging protocol is measured using an application called *tv_sim*. *tv_sim* requests tiles at a user-specified rate from the server and calculates the total number of tiles it receives. *tv_sim* can also make requests for a specified set of tiles only. This enables two *tv_sim* clients running simultaneously on a host to request the same set of tiles. Over the traditional networking path, both requests and the tiles received are duplicated over the route. The request merging protocol implemented in the active network reduces the bandwidth demand over the wireless link in half by merging requests for the same tile that occur together. We calculate performance by finding the total combined throughput of both applications in terms of tiles received per minute. We consider only the extreme case in which all duplicated requests arrive simultaneously. We see from Fig. 5 that the total combined throughput seen by the applications is better over the prototype despite its poor baseline performance. In fact, because only a single copy of every tile is returned over the wireless link and then duplicated, at request rates of 96 and 120 the aggregate throughput over the prototype is significantly larger than that over the traditional network. We also observe that at a request rate of 160 tiles/min, the performance improvement is insignificant. We suspect that the drop in throughput at that request rate is an artifact of our implementation.

If we compare the throughput curves for the traditional network from Fig. 3 and Fig. 5, we observe that the aggregate throughput is equal to the throughput of a standalone application. This validates the fact both applications compete for network bandwidth and therefore neither achieves significant throughput. However, for active networking, the aggregate throughput is twice that of a standalone application. This demonstrates that application-specific merging protocols help in reducing bandwidth demand while keeping throughput rates matching the applications needs.

VII. Related Work

The ANTS [17] prototype developed at MIT has an architecture similar to our implementation. Both implementations use the Java virtual machine as the execution environment on active nodes and transfer SmartPackets (which they call capsules) between active nodes using UDP. There are some other efforts investigating active networks. The SwitchWare project [18], a joint project between the University of Pennsylvania and Bellcore Communications, has developed a software switch on which

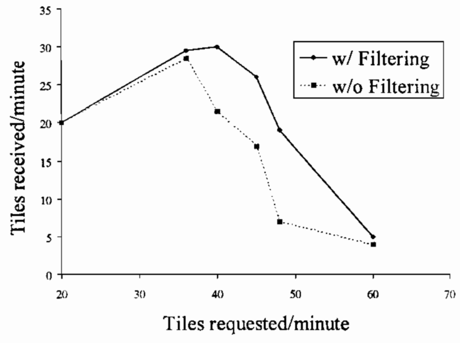


Fig. 4. Comparative throughput performance for active filtering

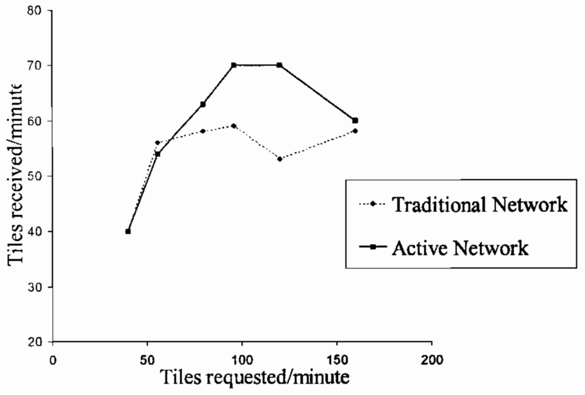


Fig. 5. Comparative throughput performance for active merging

programs can be downloaded that modify link behavior. Researchers there have also developed a new language called PLAN [19] written specifically for active networking platforms. The idea behind the Netscript project [20] at Columbia University is to provide a scripting language for creating data-flow paths in the network.

The Liquid Software project [21] at the University of Arizona investigates technologies that allow use of mobile code inside the network. The active networking project at Georgia Tech [22] embeds vendor-defined functions at network nodes. These functions are available to the user for performing user-specific services. Data packets carry control information including identifiers to select certain functions and a set of arguments describing state information to be used in the computation. This is a functional model of an active node rather than the Turing Machine model followed by our implementation. The functional model has the advantage of tighter security and better damage control but it lacks the flexibility of the latter because the vendor cannot anticipate all the functionality that users may require in the future.

VIII. SUMMARY

In this paper, we studied various kinds of services that applications require from the network and classified them into protocol classes based on common characteristics. We demonstrated that by using the methodology developed for the classes, we could solve problems in wireless networks by building new protocols and deploying them rapidly using the active network infrastructure. We also conducted studies on the performance of the protocols and showed some performance results. In conclusion, we can say that the results of the study were quite encouraging. We feel that there is a need for further study of protocol classes. The methodology outlined here, while not yet complete, provides an initial step.

REFERENCES

[1] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *Computer Communication Review*, 1996.

[2] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80-86. January 1997.

[3] Amit Kulkarni, Gary Minden, Victor Frost, and Joseph Evans, "An active network architecture for ATM WANs," *Proc. of the 3rd Mobile Multimedia Conference*, Princeton, New Jersey, September 1996.

[4] A. B. Kulkarni, G. J. Minden, R. Hill, Y. Wijata, F. Wahhab, S. Sheth, A. Gopinath, H. Pindi and A. Nagarajan, "Implementation of a prototype active network," *IEEE Conference on Open Architectures and Network Programming*, San Francisco, April 1998.

[5] D. Bakin, W. Marcus, A. McAuley, T. Raleigh, "An FEC booster for UDP Applications over terrestrial and satellite wireless networks," *Intl. Satellite Mobile Conference*, Pasadena, CA, June 19, 1997.

[6] L. Zhang, S. Floyd, Van Jacobson, S. Michel, K. Nguyen and A. Rosenstein, "Adaptive web caching," in *Third Intl. Caching Workshop*, Manchester, U.K, June 1998.

[7] E. Lindsley, *Narrowband ATM Networks*, B.S.C.S Thesis, Dept. of Electrical Engg. and Computer Science, University of Kansas, 1994.

[8] N. Yeadon, *Quality of Service for Multimedia Communications*. Ph.D. Thesis, Lancaster Univ., May 1996.

[9] Ulana Legedza, David Wetherall and John Guttag, "Improving the performance of distributed applications using active networks," *IEEE INFOCOM*, San Francisco, April 1998.

[10] Norman C. Hutchinson and Larry L. Peterson, "The x-Kernel: An architecture for implementing network protocols," *IEEE Transactions on Software Engineering*, vol. 17, no. 1, pp. 64-76, January 1991.

[11] Robbert van Renesse, Kenneth Birman, Roy Friedman, Mark Hayden and David Karr, "A framework for protocol composition," in *Proc. of the Principles of Distributed Computing*, August 1995.

[12] J. Gosling and H. McGilton, *The Java Language Environment: A White Paper*, Sun Microsystems, 1995.

[13] T. Imielinski, and J. Navas, "Geographic addressing and routing," in *Proc. of the Third ACM/IEEE International Conference on Mobile Computing and Networking*, Budapest, Hungary, September 1997.

[14] J. Abela, *Universal Format for Logger Messages*, Internet draft ftp://ds.internic.net/internet-drafts/draft-abela-ulm-02.txt, July 1997.

[15] B. R. Badrinath and Pradeep Sudame, "To send or not to send: Implementing deferred transmission in a mobile host," *Proc. of the 16th Intl. Conference on Distributed Computing Systems*, Hongkong, 1996.

[16] Prodeep Sudame and B. R. Badrinath, "On providing support for protocol adaptation in wireless mobile networks," unpublished, 1998.

[17] David Wetherall, J. Guttag and D. L. Tennenhouse, "ANTS: A toolkit for building and dynamically deploying network protocols," *IEEE Conference on Open Architectures and Network Programming*, San Francisco, April 1998.

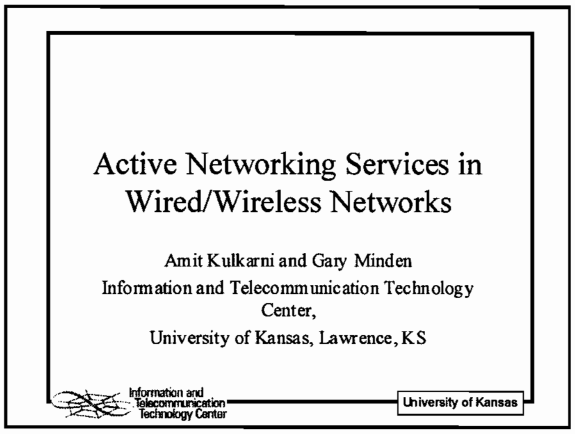
[18] Scott Alexander et al., "The Switchware active network architecture," *IEEE Network Special Issue on Active and Controllable Networks*, vol. 12 no. 3, pp. 29 – 36, 1998.

[19] Mike Hicks, Pankaj Kakkar, Jonathan Moore, Scott Nettles and Carl Gunter, "PLAN: A packet language for active networks," in *Proc. of the Third ACM SIGPLAN International Conference on Functional Programming*, pp. 86-93, 1998.

[20] Y. Yemini and S. da Silva, "Towards programmable networks," *IFIP/IEEE Intl. Workshop on Distributed Systems*, L'Aquila, Italy, October 1996.

[21] J. Hartman, U. Manber, L. Peterson and T. Proebsting, *Liquid Software: A New Paradigm for Networked Systems*, Technical Report TR96-11, Dept. of Computer Science, Univ. of Arizona, 1996.

[22] Samrat Bhattacharjee, Kenneth L. Calvert and Ellen W. Zegura, "An architecture for active networking," *High Performance Networking*, White Plains, NY, April 1997.

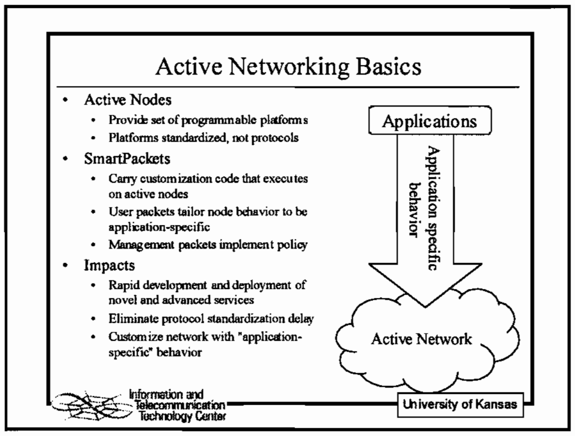


Active Networking Services in Wired/Wireless Networks

Amit Kulkarni and Gary Minden
Information and Telecommunication Technology Center,
University of Kansas, Lawrence, KS

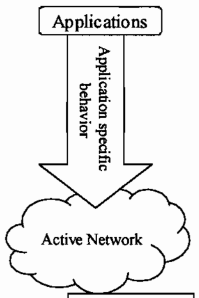
Information and Telecommunication Technology Center

University of Kansas



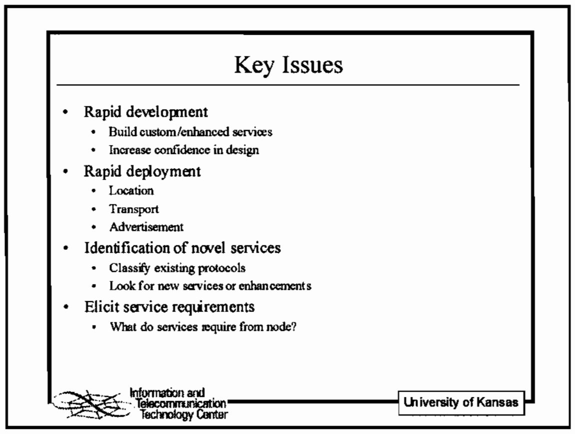
Active Networking Basics

- Active Nodes
  - Provide set of programmable platforms
  - Platforms standardized, not protocols
- SmartPackets
  - Carry customization code that executes on active nodes
  - User packets tailor node behavior to be application-specific
  - Management packets implement policy
- Impacts
  - Rapid development and deployment of novel and advanced services
  - Eliminate protocol standardization delay
  - Customize network with "application-specific" behavior



Information and Telecommunication Technology Center

University of Kansas

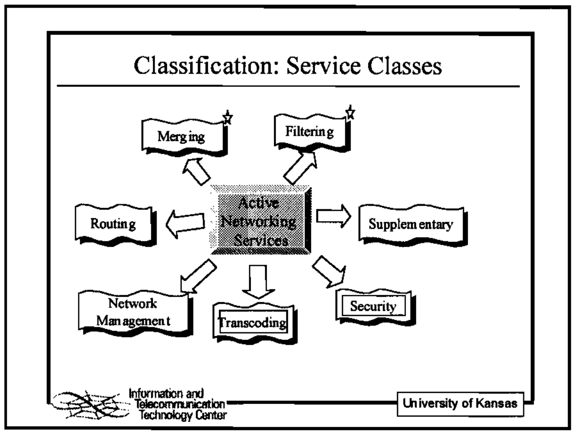


Key Issues

- Rapid development
  - Build custom/enhanced services
  - Increase confidence in design
- Rapid deployment
  - Location
  - Transport
  - Advertisement
- Identification of novel services
  - Classify existing protocols
  - Look for new services or enhancements
- Elicit service requirements
  - What do services require from node?

Information and Telecommunication Technology Center

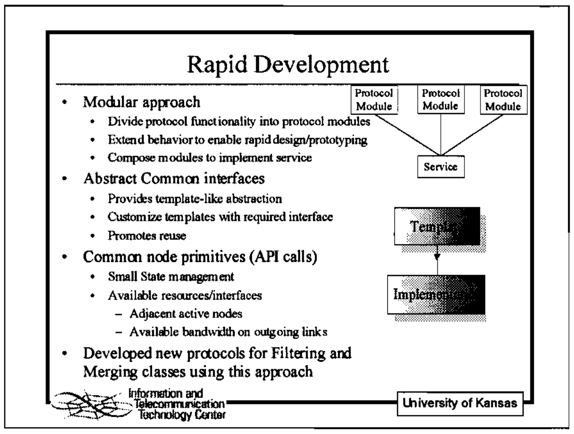
University of Kansas



Classification: Service Classes

Information and Telecommunication Technology Center

University of Kansas

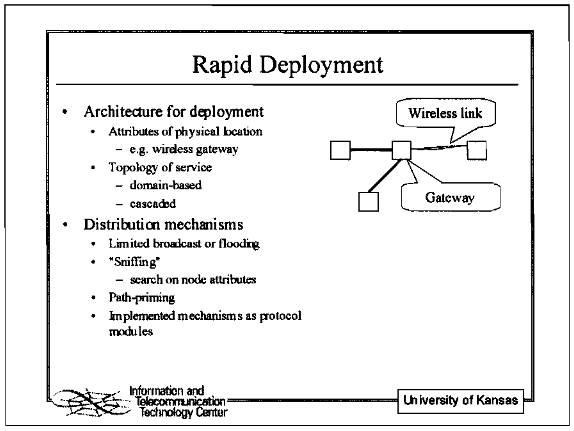


Rapid Development

- Modular approach
  - Divide protocol functionality into protocol modules
  - Extend behavior to enable rapid design/prototyping
  - Compose modules to implement service
- Abstract Common interfaces
  - Provides template-like abstraction
  - Customize templates with required interface
  - Promotes reuse
- Common node primitives (API calls)
  - Small State management
  - Available resources/interfaces
    - Adjacent active nodes
    - Available bandwidth on outgoing links
- Developed new protocols for Filtering and Merging classes using this approach

Information and Telecommunication Technology Center

University of Kansas



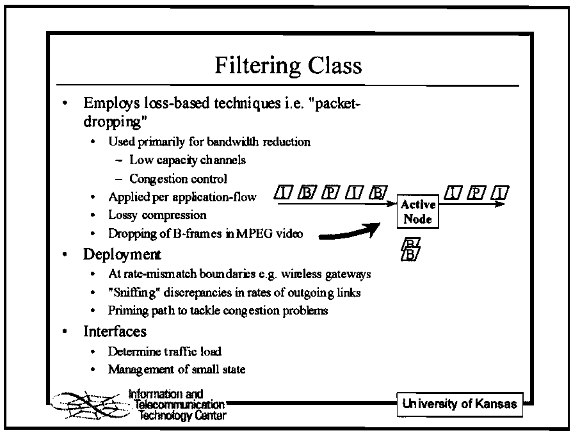
Rapid Deployment

- Architecture for deployment
  - Attributes of physical location
    - e.g. wireless gateway
  - Topology of service
    - domain-based
    - cascaded
- Distribution mechanisms
  - Limited broadcast or flooding
  - "Sniffing"
    - search on node attributes
  - Path-priming
  - Implemented mechanisms as protocol modules

Wireless link
Gateway

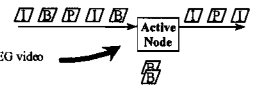
Information and Telecommunication Technology Center

University of Kansas

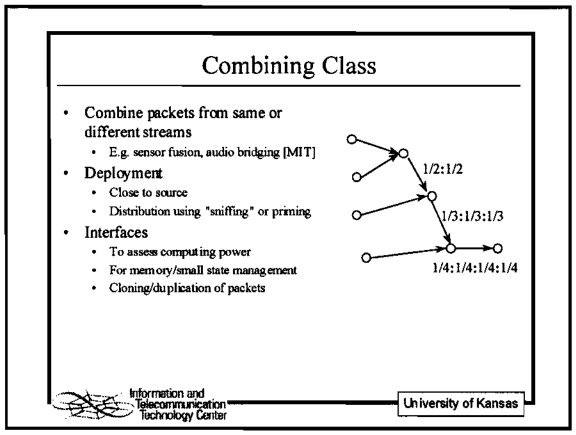


Filtering Class

- Employs loss-based techniques i.e. "packet-dropping"
  - Used primarily for bandwidth reduction
    - Low capacity channels
    - Congestion control
  - Applied per application-flow
  - Lossy compression
  - Dropping of B-frames in MPEG video
- Deployment
  - At rate-mismatch boundaries e.g. wireless gateways
  - "Sniffing" discrepancies in rates of outgoing links
  - Priming path to tackle congestion problems
- Interfaces
  - Determine traffic load
  - Management of small state

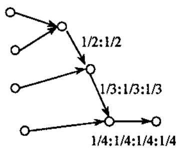


Information and Telecommunication Technology Center — University of Kansas

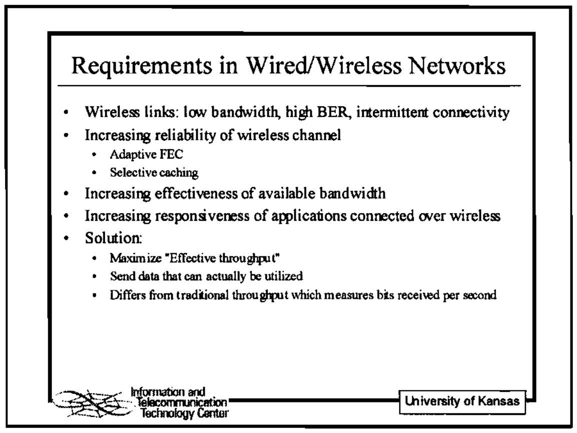


Combining Class

- Combine packets from same or different streams
  - E.g. sensor fusion, audio bridging [MIT]
- Deployment
  - Close to source
  - Distribution using "sniffing" or priming
- Interfaces
  - To assess computing power
  - For memory/small state management
  - Cloning/duplication of packets



Information and Telecommunication Technology Center — University of Kansas



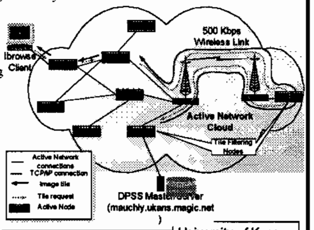
Requirements in Wired/Wireless Networks

- Wireless links: low bandwidth, high BER, intermittent connectivity
- Increasing reliability of wireless channel
  - Adaptive FEC
  - Selective caching
- Increasing effectiveness of available bandwidth
- Increasing responsiveness of applications connected over wireless
- Solution:
  - Maximize "Effective throughput"
  - Send data that can actually be utilized
  - Differs from traditional throughput which measures bits received per second

Information and Telecommunication Technology Center — University of Kansas

Active Tile Filtering

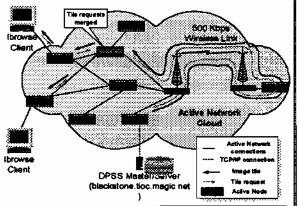
- Derived from **Filtering** Class
- Used with terrain visualization application called Ibrowse
  - Performs congestion control at the wired/wireless gateway
  - Provides rate matching for latency-sensitive data
- Node primitives:
  - Small State management
  - Bandwidth monitoring
- Deployment
  - At wired/wireless gateway
  - Location is "sniffed" out



Information and Telecommunication Technology Center — University of Kansas

Active Request Merging

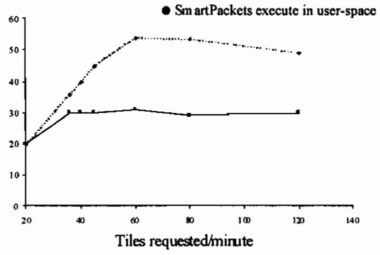
- Derived from **Combining** Class
- Consolidate duplicated requests from multiple clients
- Send request to server and multicast reply back to clients
- N-fold savings of bandwidth … n = #simultaneous requests
- Deployment:
  - Domain-based
  - Limited broadcast for request gathering
  - Cascaded
- Primitives:
  - Small State management
  - Cloning/duplication



Information and Telecommunication Technology Center — University of Kansas

Performance (Baseline)

- AN baseline performance 1.7 times as slow
  - AN execution environment is JVM interpreter
  - SmartPackets execute in user-space

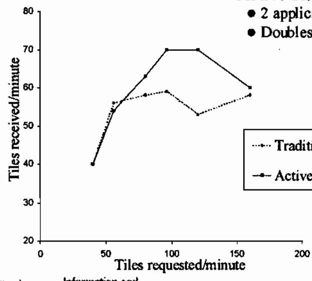


Information and Telecommunication Technology Center — University of Kansas

Application Performance

Active Request Merging

- 2 applications running Ibrowse
- Doubles "effective" capacity



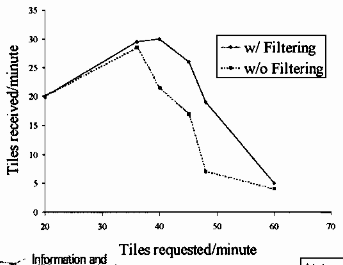
Information and Telecommunication Technology Center

University of Kansas

Application Performance

Active Tile Filtering

- Improves performance by ~ 30 %



Information and Telecommunication Technology Center

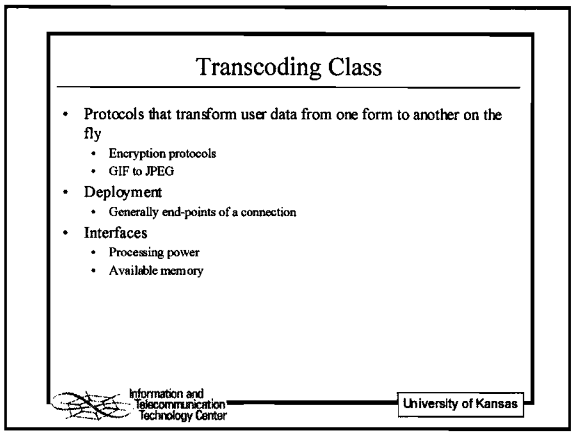
University of Kansas

Conclusions & Future Work

- Classification enables creation of templates for rapid development of new protocols and services
- Architecture deployment methods captured as modules
- Demonstrated use and performance advantage with real application
- Future work:
  - Define protocol module behavior and interfaces
  - Verify module composition to make statements about service

Information and Telecommunication Technology Center

University of Kansas

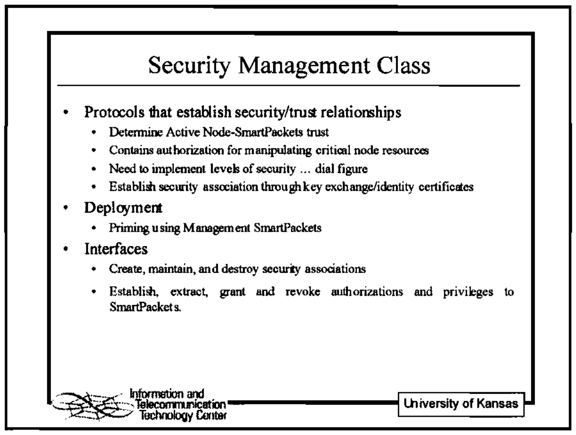


Transcoding Class

- Protocols that transform user data from one form to another on the fly
  - Encryption protocols
  - GIF to JPEG
- Deployment
  - Generally end-points of a connection
- Interfaces
  - Processing power
  - Available memory

Information and Telecommunication Technology Center

University of Kansas

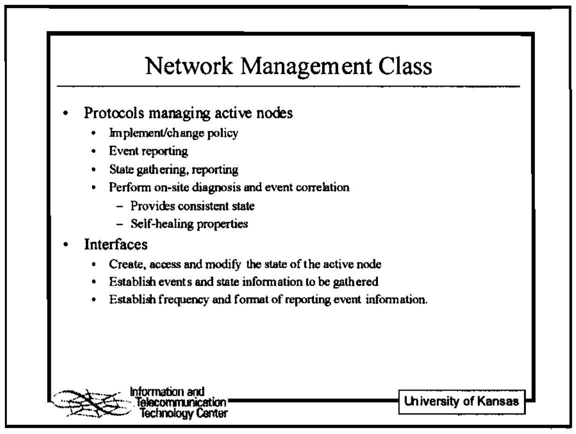


Security Management Class

- Protocols that establish security/trust relationships
  - Determine Active Node-SmartPackets trust
  - Contains authorization for manipulating critical node resources
  - Need to implement levels of security ... dial figure
  - Establish security association through key exchange/identity certificates
- Deployment
  - Priming using Management SmartPackets
- Interfaces
  - Create, maintain, and destroy security associations
  - Establish, extract, grant and revoke authorizations and privileges to SmartPackets.

Information and Telecommunication Technology Center

University of Kansas

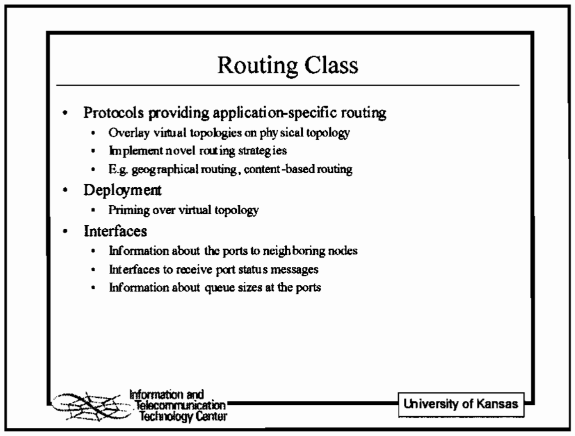


Network Management Class

- Protocols managing active nodes
  - Implement/change policy
  - Event reporting
  - State gathering, reporting
  - Perform on-site diagnosis and event correlation
    - Provides consistent state
    - Self-healing properties
- Interfaces
  - Create, access and modify the state of the active node
  - Establish events and state information to be gathered
  - Establish frequency and format of reporting event information.

Information and Telecommunication Technology Center

University of Kansas

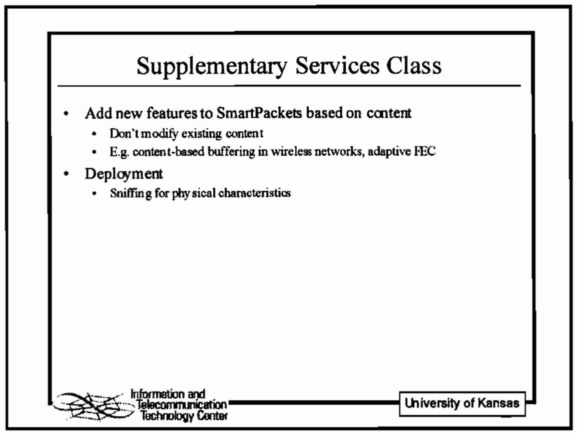


Routing Class

- Protocols providing application-specific routing
  - Overlay virtual topologies on physical topology
  - Implement novel routing strategies
  - E.g. geographical routing, content-based routing
- Deployment
  - Priming over virtual topology
- Interfaces
  - Information about the ports to neighboring nodes
  - Interfaces to receive port status messages
  - Information about queue sizes at the ports

Information and Telecommunication Technology Center

University of Kansas



Supplementary Services Class

- Add new features to SmartPackets based on content
  - Don't modify existing content
  - E.g. content-based buffering in wireless networks, adaptive FEC
- Deployment
  - Sniffing for physical characteristics

Information and Telecommunication Technology Center

University of Kansas