Panda: Providing the Benefits of Active Network to Legacy Applications

Kevin Eustice, Vincent Ferrerio, Richard Guy, V. Ramakrishna, Alexey Rudenko, and Peter Reiher

Abstract

1. Introduction

The continuing improvements in accessibility, speed, and coverage of various kinds of computer networks has lead to users relying heavily on connectivity for normal business. However, the widely varying characteristics of networks often cause problems for their use, since applications typically assume some minimal quality of service from the network. If the network in its current state cannot provide that quality, many applications work poorly or not at all.

In many cases, more intelligent handling of data in the network could ameliorate these problems and allow applications to work well even under difficult network conditions. Active networks offer this promise by allowing substantial programmability of the network. However, most existing active network systems work on the assumption that new applications are written that explicitly instruct the network on how to handle their data streams. This approach offers no benefits to applications that were written before active networks were created, nor to later applications that were not written with the possibilities offered by active networks in mind. Even applications that were written for active networks are limited in their use of them to the creativity and foresight of the application designer, who must become not only an expert in his own application area, but in networking, to make effective use of the active network.

Panda is a middleware system that provides the benefits of active networks to unaware applications. Panda traps data streams from those applications, converts them to active network packets, determines the network conditions, makes a plan of which adaptations to apply to the packets to meet prevailing conditions, and deploys the code necessary to ensure proper handling of the packets. Panda is transparent to the applications it services, though of course any permanent alterations it makes in the data stream will be visible at the destination.

This paper describes the basic architecture and current implementation of the Panda system. The paper also describes demonstrations of the efficacy of Panda and presents performance data on the system. It discusses the lessons learned during the Panda project about transparent adaptation of data streams, composition of multiple adapters, and automated planning for active networks.

2. Panda Architecture

To ease implementation, Panda is built on top of an existing Active Networks execution environment (EE). This EE provides Panda with basic active networking services, such as executing code at a node on behalf of a packet, deploying adaptation code to the required nodes in the network, etc. Panda is implemented on top of the ANTS execution environment [Wetherall98]. ANTS is a Java toolkit that provides a protocol-based programming model for customizing packet forwarding through a network. While ANTS did not perfectly match the Panda model of active networks, it required only minor alterations to support Panda.

Panda currently supports UDP applications. The underlying ANTS system is capsule-based and makes no guarantees regarding the delivery of capsules or the order

that capsules will be received at the destination, much like UDP. Also, multimedia applications, which tend to use UDP, are good candidates to benefit from a distributed adaptation system.

Currently, Panda supports unicast applications only, although it has been used for simple multicast-like operations like forwarding incoming data to two different outgoing branches.

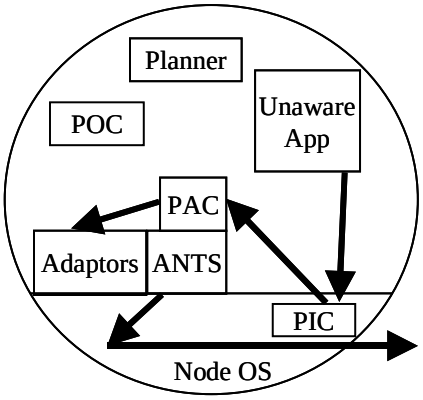


Figure 1. The Panda Architecture

The Panda architecture has four modules, each of which addresses a major task in the middleware system (Figure 1). The Panda Interception Component, or PIC, is responsible for obtaining data from clients. The Panda Adaptation Component, or PAC, deploys and runs adapters for multiple client applications. The Planner determines a set of adapters that solve the network limitations to meet the users requirements and preferences. The Panda Observation Component, or POC, deals with gathering and reporting information required for all other Panda components, including planning.

Panda does not address all relevant issues for an active network middleware component, since the priority was to demonstrate the feasibility of the idea. First, Panda uses only ANTS' mechanism for code transport, which is not ideal for its purposes. Second, Panda does not address any security issues involved in providing a distributed adaptation service, though associated research [Li02] has addressed some important security issues. Third, since Panda works with UDP streams, it does not provide any sort of reliability, though again associated research [Yarvis00] addresses these issues.

Finally, Panda does nothing with routing, though alternate routing policies could be beneficial.

The Panda Interception Component, or PIC, must intercept all data streams that Panda may wish to handle. Depending on the facilities provided by the host operating system, this interception can be accomplished in different ways. The current implementation uses a Linux loadable kernel module to intercept socket calls. The firewalling capabilities built in the Linux OS could also allow the necessary redirection and masquerading of connections. Linux iptables could handle this problem. Systems like the x-kernel [Hutchinson91] and Scout [Mosberger96] have built-in capabilities to control handling of network connections. The PIC must also have some mechanism to instruct it about which data streams to intercept.

The PAC is the core of the Panda system. The PAC installs the necessary adapters for a data stream, delivers capsules to the proper adapters, and generally controls the flow of a data stream through Panda nodes. Because these responsibilities heavily overlap the typical behavior of an execution environment, this portion of Panda is tightly coupled to the underlying EE, ANTS in the current implementation.

Panda adapters are modules that accept a data packet and can perform arbitrary modifications on that packet, including dropping it or converting it into more than one packet. Panda may deploy more than one adapter for a single data stream on a particular node, so the system must allow for the output of one adapter to serve as the input for the next. Since the packet can be dropped, Panda must also allow for situations where not all adapters deployed on a node are actually invoked to handle a particular packet.

During execution, an adapter may store data at several different locations in the Panda environment. The ANTS node cache and the POC provide interfaces to access many distinct data items. The Panda system also provides an additional interface to dynamically store data within the capsule, known as the capsule cache. The content of the capsule cache is maintained as the capsule traverses the network and is available to any adapter that runs on this capsule instance.

The Panda Observation Component, or POC can be viewed as the central service for messaging between all Panda components, analogous to a CORBA ORB. A typical Panda node has a POC running locally. Two types of components connect to the POC: sensors and clients. Sensors generate information. Clients obtain the data generated by the sensor via the POC. In some cases a program may be both a client and a sensor to the POC; for example, a program that provides hysteresis-type functions on data to a client could obtain the original data from another POC sensor.

The Planner is the most important client to the POC in the Panda system. The Planner uses to POC to determine the current network conditions and other information needed to determine a suitable plan for an application's data stream. The Planner also needs user preferences so it can tailor the plan to suit a particular user's needs. User preferences can be implemented as a POC sensor that interacts with the user, and this configuration reduces the complexity of the Planner as it only needs to be a POC client to obtain this additional information regarding the user.

Panda is capable of supporting multiple different planners. Initially, Panda used a very simple template-based planner. This simple planner has been replaced by a far more powerful planner based on heurisic search [insert reference to Panda planning]. In brief,

this planner uses information about the data stream, network and node conditions, and adaptor availability to search the space of all possible plans for the best plan. Heuristics based on constraints of adaptation and observations of how adaptation should be deployed allow the planner to create high quality plans in much less time than an exhaustive search would require. Planning runs on the node that initiates the data stream.

Panda, under normal conditions, will work in a transparent fashion using automated planning; the application programmer or a user need not know anything about it. This may not be desirable in all situations. An application may be aware of the active network; it may have better knowledge of network and system conditions. Therefore it makes sense to allow an application to have the option of taking part in the planning process for adapter deployment.

While Panda is designed to operate without user or application assistance, such assistance could allow better adaptation of data streams, when it is available. Panda thus gives the application programmer a standard socket API to control Panda operations, for sockets controlled by Panda. Low-level functions like packet interception and socket proxying are done by Panda; the applications get a higher-level view of the network. The API allows the applications to control the planning process. The application may disable Panda from performing its planning, or it could reject the plan in favor of its own. Panda provides finer mechanisms for influencing planning, as well.

Panda also provides a user interface so that users can set preferences for how Panda will handle their data streams. Users have the option of selecting which streams and data types to adapt and with what priority. Voice transmission may have higher priority than bulk data transfer, for example. Users can choose data fidelity levels, such as minimum

tolerated image resolution. Other options include security level desired and communication delay constraints. All these preferences are used as input by Panda when it performs its automated planning.

There are other interface features that are not directly related to Panda. The application will be provided APIs to communicate with the system in order to obtain the latest information about the system and network conditions. Such information, being critical to performance, can be communicated asynchronously in the form of events to the application to trigger replanning.

3. Panda Implementation

3.1. Basic Implementation Details

The current Panda system has implantations of the PIC, PAC, and Planning components, in addition to various adapters. The POC is under development. Panda is written in Java, with the exception of the PIC, which contains a Linux loadable kernel module and a JNI interface to control its operation. The PIC and PAC contain approximately nine thousand lines of code, not including code for adapters. **[Alexey to provide Planner statistics]**

Panda is built on top of a modified version of the ANTS 1.2 distribution. The most significant change to ANTS was to support larger capsules – larger in both size of code and size of the data sent over the network. Additionally, Panda required changes to the ANTS dynamic code-loading system to allow capsule code to be loaded from any node and run from any capsule. The latter changes break the fundamental principles of how the ANTS system works, but these changes are not necessary to run Panda.

Panda runs on the Linux operating system with kernel from the 2.0 or 2.2 series. It requires a JVM version 1.1 or higher. It has also run on Janos, using a customized version of the Kaffe VM. The kernel module of the PIC needed to be reimplemented to work in the Janos environment, but the Java interface to the PIC remained the same, only requiring Java code changes to support two different interception implementations.

3.2. PIC Implementation

The current Panda PIC is an LKM stacked on top of the native networking functions to provide additional control over the proxy and masquerading facilities built into Linux. Using a kernel module for interception allows Panda to intercept any application's data stream running on the node, regardless of how the application is linked or what libraries it uses. Panda receives an application's data at the system call level before any network-level transformations have occurred, like segmentation or the addition of checksums. Unfortunately, this approach is subject to any user-level buffering that may occur when using standard I/O libraries. Panda also has no access to any information that is present in a user-level networking interface, if one is used.

In the case of UDP communications, the middleware opens a new UDP socket for interception and performs a LKM sockopt() informing the LKM that this socket wishes to intercept certain UDP packets. The LKM diverts any outgoing datagram that matches the intercept description from the original destination to the interception UDP socket opened by the middleware service by changing the destination address of the packet before it reaches the normal kernel networking code. The original destination address is stored in the module in a per-socket data structure. After receiving a diverted datagram on the

interception socket, the middleware service issues an LKM sockopt() to obtain the packet's original destination address. At this point, the middleware is now able to send the payload over the active network.

At the destination Panda sends a datagram to the real destination application, but uses the LKM to masquerade as the original source. As in packet interception, the middleware makes use of a LKM sockopt() to control the masquerade address for the packet. The middleware sends the packet over a socket, and the LKM in turn makes use of facilities in the standard Linux kernel networking code to perform masquerading on the packet.

UDP communication is connectionless, so it is unnecessary for an application to send a close signal over the network to another computer. Without a close signal, the Panda system cannot reliably determine when to free any resources associated with a data flow. To solve this problem, the LKM watches for UDP socket closes and sends a close signal to any interception socket that has intercepted data from the closing socket.

Interception is initially performed on UDP packets or TCP connections destined for well-known port numbers. Since most applications make use of well-known port numbers to reach standard services on a server, this has not proved to be a limitation. While this approach is certainly less flexible than interception based on signatures that may be found in the data stream itself, it incurs less overhead and latency to the applications that cannot receive benefit by the middleware service.

Interception can also occur on other packets or connections that are related to the application but not on a well known port number. For instance, in a TFTP file transfer, only the initial file request is sent to a well-known port number; the data transfer and acknowledgement packets are sent to operating system-assigned port numbers. In these

cases, the new port number to intercept can be determined from the source address or from information in the payload.

3.3. PAC Implementation

. The PAC is implemented as an ANTS application that handles data from multiple user applications and converts the data into capsules that are sent over the active network. At the destination, the PAC removes the data from the capsule and delivers it to the receiving application. The design of ANTS does not require Panda data streams to pass through the PAC at intermediate nodes.

3.4. Panda Adapter Implementation

Adapters in the Panda system are placed in a special method of an ANTS capsule, with one adapter per capsule type. This placement provides a number of benefits and also allows reuse of much existing capsule code with a minimum of changes. One of these benefits is that the loading of capsule code to a node is handled by the ANTS system. Additionally, Panda benefits from any capsule-code security mechanisms that are built into ANTS when loading capsules at a node.

In Panda, adapters have complete control over the capsule, including routing and transformation. Panda is designed to provide as much flexibility in the adapters it can use as possible. This decision also reduces the size and complexity of the Panda code resident in the capsule by delegating routing and forwarding to an adapter.

Panda creates a plan of which adapters to deploy to allow the data capsules to reach their destination and receive the special treatment required by current network conditions. When a Panda capsule begins evaluation at a node, it does not know what adapters need to be run. The *plan access method* determines which adapters a capsule should run. To

support different styles of planning, there are 3 plan access methods built into Panda. First, the plan could be embedded into the capsule. Second, the plan could be in the ANTS node cache. (This method is used for Panda's heuristic-based planner.) Finally, the capsule can visit the planner on the current node to determine the set of adapters to run there. A capsule may try any combination of these plan access methods, depending on how the capsule was initialized. Should all of these methods fail to provide a set of adapters to run, as in the case where a capsule is forwarded along an unexpected link, a simple shortest-path forwarding routine built into the data capsule is run.

Once a set of adapters is found at a node, control of execution is transferred to the first adapter, which has complete control over the capsule. It may choose to transform the payload or headers (including the planning information), forward the capsule, or run the next adapter. The list of adapters to run is kept in memory, and the currently executing adapter can either call the next adapter in the list or terminate execution of the capsule after it has performed its functions. Most adapters will simply call the next adapter on the list until the end of the list is reached, where capsule execution will terminate. This includes forwarding/routing adapters, which should be normally placed at the end of the list of adapters to run.

3.5. POC Implementation

The POC must accept sensor information from various sensors, including ones that do not reside on the local node. To allow for different types of POC sensors to be built, the POC employs a common modular interface to add and query sensors. This modular interface maps neatly into the JAR and Interface features of the Java system. This system can also integrate with existing monitoring systems, as the POC sensor module can

simply act as a bridge between the POC and the component that performs the actual monitoring.

Clients to the POC are typically other Panda components. POC clients can determine the available sensors, add and remove sensors, and obtain information from a sensor attached to the POC. Adapters can act as either sensors or clients of the POC, although because adapters are implemented as capsules, they cannot communicate with the POC without special provisions. For operations where the data is not time-sensitive, the client can get POC information and store information as a POC sensor in the ANTS node cache. Periodically, the PAC will examine the contents of the node cache and act as a proxy to the POC for the adapters. This method of communication with the POC lessens the amount of time the adapter spends performing its role as a sensor or client. The adapter also has the ability to communicate with the POC through the use of an ANTS extension. After finding the POC extension on a node, an adapter acts as any other client or sensor to the POC.

POC clients usually run on the same node as the POC. However, many clients, such as the Planner, need access to information that resides on other nodes. Thus, the POC implements a gateway module to query information that resides on a remote POC. With the module, a client asks its local POC for information residing on a remote POC, and the gateway module obtains the information from the remote POC transparently to the client on the local machine. The gateway module can be implemented as a standard client and server to the local POC that runs on all nodes.

3.6. Panda Planner Implementation

The Panda planner runs a simple protocol to gather all information necessary to build its plan. However, doing so and performing the heuristic search can take some time. Therefore, Panda also creates a temporary plan quickly, to allow data to start flowing before the normal planning procedure completes. This temporary plan is built on a per-node basis, with each node using purely local information from itself and the next Panda node to determine which adapters to deploy on those nodes. These temporary plans can be very far from optimal, but they allow some data to flow while the full planning procedure occurs.

3.7. Sample Panda Applications

An early application of Panda assisted in transmitting a video from a server to two destinations with differing link throughputs. Without Panda, the server would have to send a customized version of the video stream to each client to provide them with the maximum video fidelity attainable over their respective connections. With Panda, we used two adapters to achieve a better effect. The first adapter duplicated a single, original quality, unicast video stream from the server and forwarded them over high quality links to two intermediate nodes. The second adapter was run at these intermediate nodes and filtered the video stream to meet the throughput restrictions to the clients, who thus received a higher quality of service while reducing the throughput and computation load on the server.

A more complex application of Panda involved multiple components from UC Berkeley, the University of Utah, ISI, and Columbia. In this scenario, a Berkeley Ninja server was sending the video stream accompanying a presentation to a client connected through an overloaded link. The video stream contained multiple versions of the video,

each encoded at a different quality. Panda intercepted the video stream and performed two actions. First, it setup a Virtual Area Network from the source to the destination node, using software from Columbia. The VAN used RSVP to guarantee the throughput over the congested links. At an intermediate node running Panda and Janos, an adapter only forwarded the highest quality of version of the video stream that the client could receive.

Another demonstration of Panda also involved interoperation with UC Berkeley's Ninja, Columbia's Virtual Active Networks and Nestor, and the University of Utah's Janos system. The scenario for the demonstration was a videoconference, with two different video/audio sessions being streamed to a third participant, in an extended Y-configuration, through a heterogeneous network with a variety of problems. Network problems included a packet storm on the wired segment, as well as extensive wireless competition. In order to delivery acceptable video and audio, network conditions had to be analyzed, and the media appropriately adapted. Adaptation in this case was selective layer-based distillation of the video, encoded in the WaveVideo wavelet codec [Fankhauser99], based on prioritization of the streams. Prioritization was determined by a bandwidth analysis of the audio traffic, hypothesizing that more audio traffic would indicate a speaker. Due to the tremendous number of packets from the videoconferencing sessions, final wireless link was incapable of delivering acceptable video for both senders. Thus, Panda was required to selectively drop packets from the less desirable session, while maximizing the quality of the "focused" session. The end result was a usable video stream from one of the cameras, which switched back and forth as the speaker focus changed.

4. Panda Performance

4.1. System Overheads

Panda puts substantial code (itself, ANTS, and adapter code) in the path of packets it intercepts. The overheads associated with this code determine the domains for which use of Panda will be beneficial.

Error bars on all figures show the value of standard error, unless otherwise indicated.

One fundamental overhead is the additional latency of delivering a packet. The following method was applied to measure one-way packet latency. The packets were stamped with the local time on the source machine. Upon the arrival at the destination machine the stamped time was subtracted from the destination local time to obtain *measured time delivery*. The synchronization of the source and destination machines' clocks was done with NTP. The NTP server was located on the destination node. The source node synchronized itself to the destination local time before the first packet was sent to the destination. Then 20,000 packets were sent to the destination. After the last packet was delivered, the source machine measured the *skewing value*. It was presumed that skewing grows uniformly by time. The *actual time delivery* was calculated with formula for each data packet *n*:

$$ActualTimeDelivery(n) = measuredTimeDelivery(n) - \frac{skewingValue}{20{,}000} \bullet n$$

The connection was tested with twisted pair sequential connections of up to four computers as shown on Figure 2. Dell Inspiron laptops with 333 MHz processors were used for one set of tests and Hewlett Packard Omnibook 4150 laptops with 500 MHz processors for another set of tests; all machines used Linux Red Hat 7.0 with the 2.2.16

kernel. Xircom RealPort2 Ethernet 10/100 PCMCIA cards were used for the network connection between the machines. The source and destination machines ran a user application and the Panda code concurrently. The priority of the user application was set lower on the source machine and higher on the destination machine to ensure proper allocations of resources.

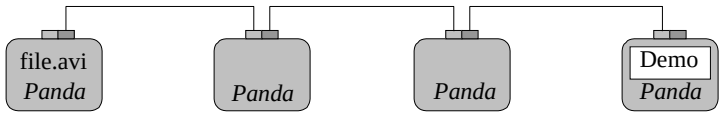


Figure 2: Panda peer-to-peer connection

Figure 2. Experimental setup

Throughput of the network links is varied among 150 Kbps, 800 Kbps, 2000 Kbps, and 5000 Kbps using CBQ.

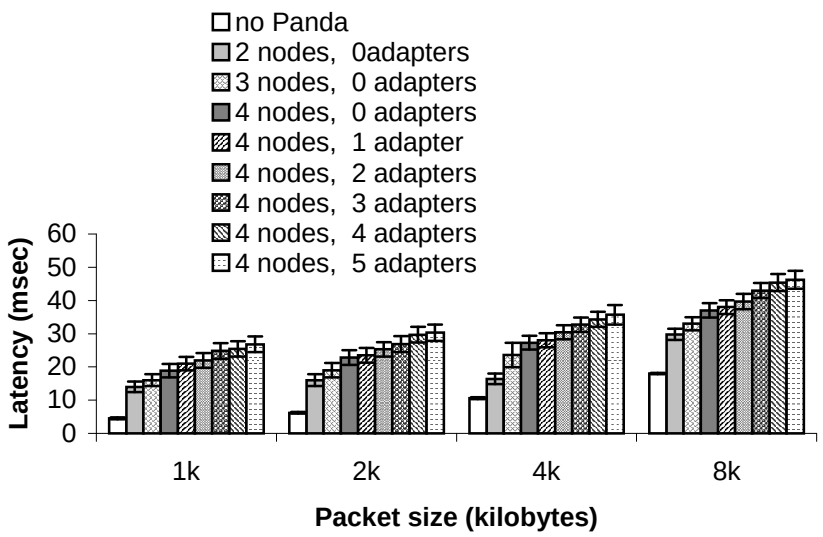
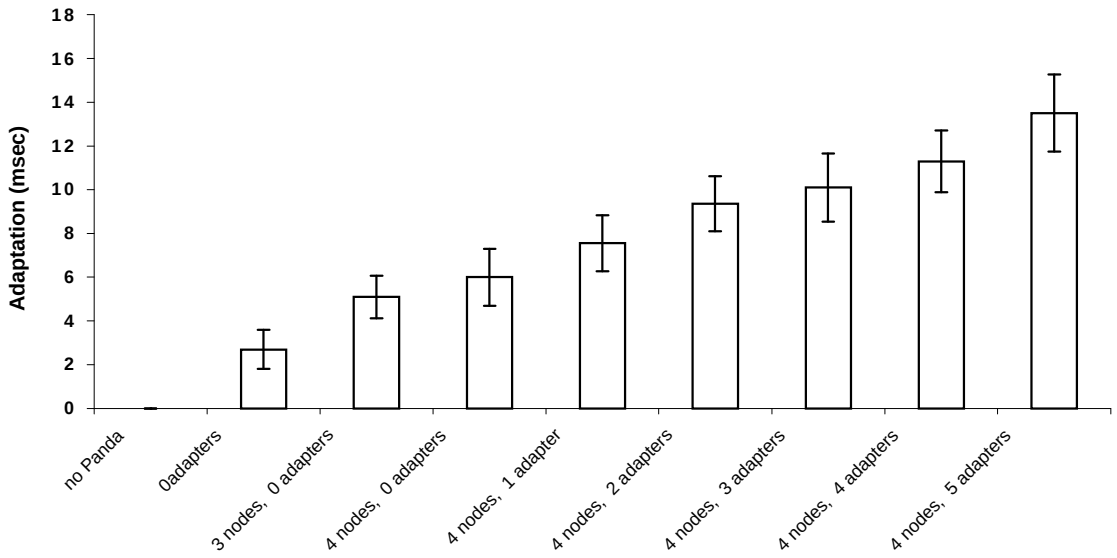


Figure 3. Packet delivery latency

Figure 3 presents packet delivery latency for different packet sizes. The packet delivery latency also contains the adaptation latency. Figure 3 shows that adding Panda to a data stream increases its latency 50-150%, with longer packets seeing less effect. Adding more Panda-enabled nodes or more adapters modestly increases the delay for each addition.

Figure 4. Null Adapter Latency

Figure 4 presents the latency of null adapters. All adapters were deployed on one of the nodes of the connection. Without Panda no adapters can be deployed, so the extra latency for that case is defined to be zero. Every Panda node always runs at least one *forward* adapter, whose only task is to forward a packet to the next node after all other adapters are executed. A number of forward adapters equal to the number of connection



nodes is always present in a Panda connection but it is not counted on these graphs.

Figure 5 presents the latency of the adaptation with real adapters. This figure and Figure 6 were obtained by running a WaveVideo application on the Panda setup shown in Figure 2, using adapters that filtered the video and/or performed encryption and decryption. Since real adaptations are often CPU bound, more powerful machines incurred less latency, as shown in Figure 5.

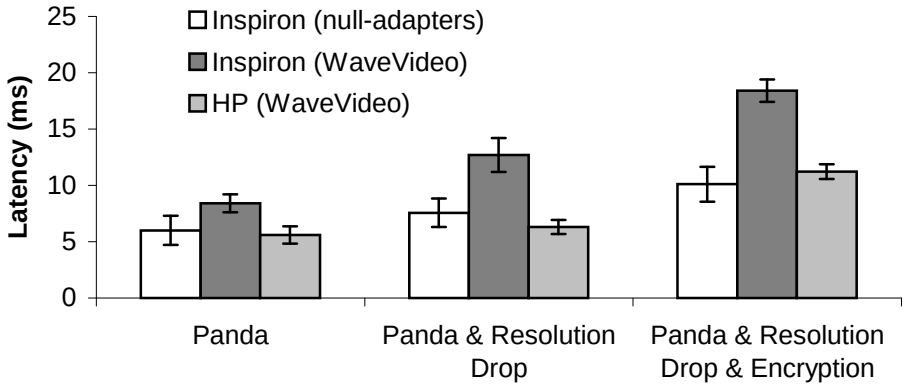


Figure 5. Latency of Running Real Adapters

Figure 6 shows how Panda throughput grows with packet size. The packet size of multimedia applications varies because some applications apply their own compressing protocols to the data packets. Error bars represent 95% confidence intervals.

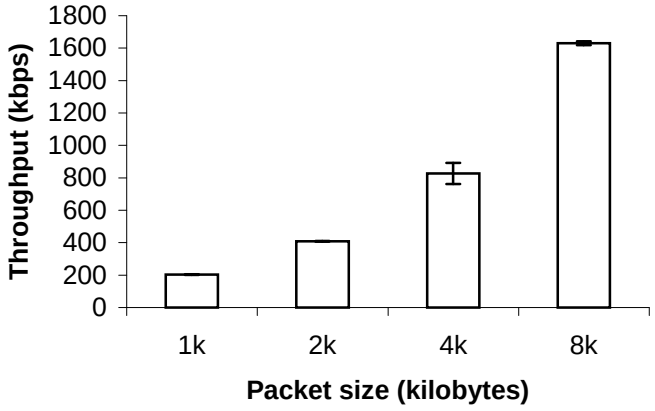


Figure 6. Panda Throughput

The planning procedure consists of planning data gathering, plan calculation, and plan deployment. Planning data gathering takes one round trip; the source node forwards

the data gathering message to the end node and waits for its return. Planning data gathering throughout four Panda nodes takes 108 +/- 2.85 milliseconds.

Figure 7 shows the latency of the plan calculation for the connection that may require no adapters, or just a Resolution Drop adapter, or both Resolution Drop and Encryptor/Decryptor adapters. The graph shows that plan calculation latency does not depend on the available throughput.

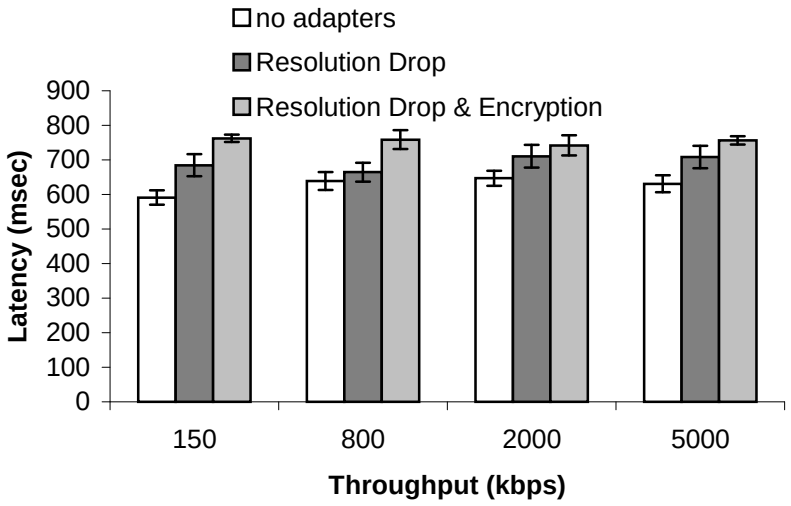


Figure 7. Plan Calculation Overhead

Figure 8 shows how the latency of deploying the adapters selected by the planner depends on adapter size and the available link bandwidth. Resolution Drop is a very small adapter that contains a few lines of code. Encryption is a heavyweight adapter that processes every character of user data. The larger the adapter, the longer it takes to deploy it. The deployment latency does not depend bandwidth unless it is less than 150kbps.

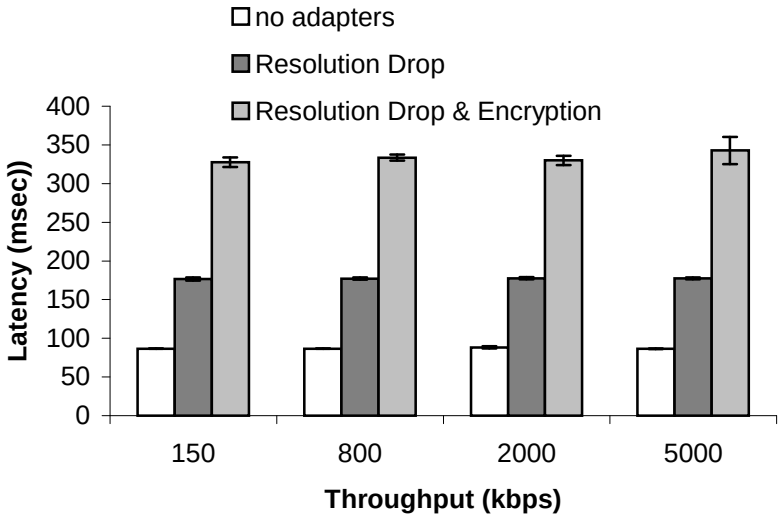


Figure 8. Plan Deployment Latency

4.2. Panda Benefits

Panda is worth using only if the benefits it offers outweigh its overheads. For some benefits, such as encryption, quantifying the benefit is hard, particularly for purposes of comparison to latency overheads. Here we present benefit metrics that are more quantifiable and take the latency overheads into account. In particular, we present improvements in the Peak Signal to Noise Ratio (PSNR) for the WaveVideo application discussed earlier. Figure 9 presents PSNR luminance on Dell Inspiron machines with a link bandwidth limited to 150 Kbps.

Without Panda, the PSNR curve declines when the channel is saturated and more or less random video packets are dropped. Panda, using the Resolution Drop adapter, intelligently adjusts to the limited bandwidth by dropping packets representing lower resolution video components. As a result, once Panda has completed its planning phase and deployed its adapters, its PSNR curve improves and exceeds the non-Panda curve. The PSNR performance of the Panda with Resolution Drop and Encryption adaptation in

some areas can be even better than the Panda with Resolution Drop only, due to fortunate buffering effects caused by the extra delay of encryption.

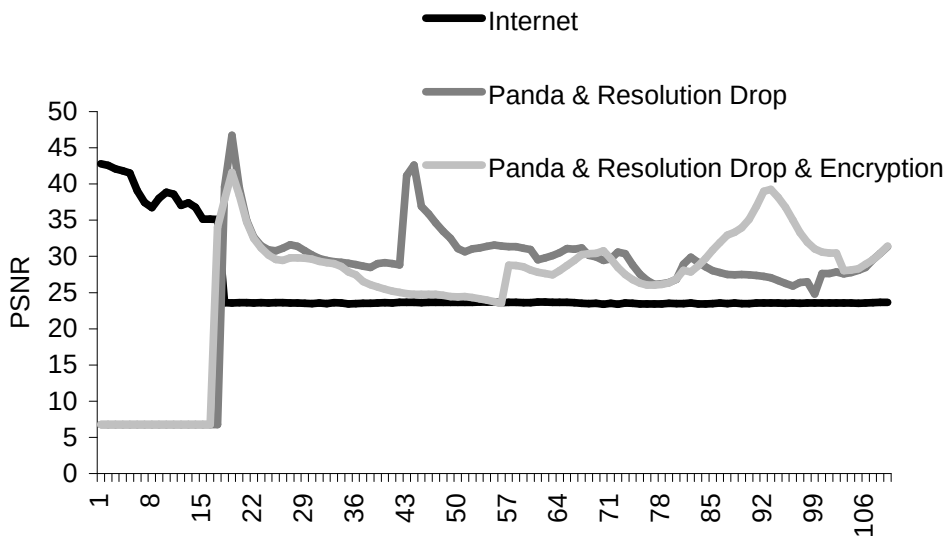


Figure 9. PSNR for WaveVideo Application

5. Related Work

Panda is the intellectual descendant of Conductor [Yarvis99]. Conductor is a TCP-based open architecture framework providing a distributed, coordinated adaptation facility. Similar to Panda, Conductor supports application transparent interception and distributed, coordinated adaptation of the network stream. Unlike Panda, Conductor offers an extensive security model, as well as a reliability model designed for adaptation called *semantic segmentation*. As Conductor is a TCP-based framework, the adaptation library for Conductor is substantially different than Panda's, focusing on HTTP, POP, and other stream based adaptations.

The Protocol Boosters [Feldmeier98] adaptation framework provides a general approach to network-level adaptation. The framework allows either a single adaptation module or a pair of modules to be transparently deployed, adding new features to existing

protocols, such as forward error correction or fast retransmission. Boosters typically provide lossless adaptation, since the system provides no support for ensuring reliable delivery if packets intended for delivery are generated, dropped, or permanently altered by a booster. Boosters are composable, but the system does not provide support for selecting a set of boosters that will perform well together. Panda substantially differs from Protocol Booster with its planning capabilities, as well as its support for lossy adaptation.

Transformer Tunnels [Sudame98] use IP tunneling to alter the behavior of a protocol over a troublesome link. Once created, a transformation function is applied to all data flowing through each tunnel. Generally, Transformer Tunnels are used to provide protocol-independent adaptations, such as consolidation of packets, scheduling of transmissions to preserve battery power, encryption, lossless compression, and buffering. Transformer Tunnels are transparent to applications and may be interoperable with application-level adaptation provided by proxies. However, no mechanism is provided to compose transformation functions or to coordinate transformations with externally provided adaptations. Panda's adaptor model allows this composability; additionally the Panda Planner coordinates various adaptations across multiple links.

Proxies are often used to handle single troublesome links, particularly links close to client nodes. One of the most advanced proxy solutions is the Berkeley proxy [Fox97]. This system uses cluster-computing technology to provide a shared proxy service for a wide variety of PDAs. The proxy can provide a variety of application-level adaptations, including transformation (changing the data from one format to another), aggregation (combining several pieces of data into one), caching, and customization (typically

converting a data format for use by a particular PDA). The Berkeley researchers have investigated methods of composing adaptations on a single machine [Gribble99]. They have also examined the use of a clustered proxy service to provide highly reliable and scalable services to a large number of customers. In particular, their proxy technology has been deployed for large-scale, real-world use, supporting palm-computer based web browsing in a metropolitan-area wireless network [Fox98]. The Berkeley Proxy and other proxy solutions typically work at a single location in the network, while Panda is designed for distributed adaptation at multiple locations.

[Add info on recent projects reported in last Openarch that are relevant to Panda.]

6. Conclusions

The Panda project has demonstrated that active network technology can be applied usefully even to applications that were not written with active networks in mind and that are not altered to work with active networks. This demonstration substantially increases the potential audience for the improvements offered by active networks. Not only are legacy applications potential users of active networks, but future programmers can concentrate on the needs of their applications, rather than the complexities of programming an active network. Where suitable, they can provide hints and direction to Panda or a similar system, but they can still expect that the active network will perform beneficial actions on their data streams even without such advice.

Panda achieves reasonable performance despite being unoptimized and running on an early version of ANTS, which is known to have poor performance. Even with these disadvantages, realistic applications receive user- and application-visible benefits from

Panda. In a more optimized form, Panda could provide greater benefits to a wider range of applications.

Panda's architecture is well suited for partial deployment of active networks. Panda must run on the source and destination node (though further development could remove even those restrictions), but otherwise does not require intermediate nodes to participate in the active network. Of course, non-participating nodes cannot perform useful adaptations, but this approach allows selective deployment of Panda at nodes that are close to troublesome links, or that often are overloaded, or that have other characteristics suggesting that they are a good spot for adaptation. The more such nodes deployed, the more options available to Panda.

Panda has also demonstrated that automated planning of active network adaptations is possible and efficient. Panda's automated facility plans sufficiently quickly to provide a plan early in most data streams, and the plans provided are often as good as those found by exhaustively testing all possibilities. Without a reasonable planning facility, the Panda approach could not be used in the real world, so this demonstration is key to its future success. Further, this result suggests that automated planning based on heuristic search or other AI techniques might have a wider applicability in solving many distributed systems problems.

A final lesson from the Panda project is that early choices can have long-lasting implications. The decision to build on an existing execution environment (rather than creating a new one) and the choice of ANTS for that EE had profound implications for the project. Much of the Panda implementation effort was spent making simple concepts fit into a framework that wasn't designed to support them. The choice had other

implications, such as mandating an early commitment to performing the work in Java. This choice was not a mistake, since the resulting system demonstrated all the hypotheses of the original project, but it did have wide ranging effects on the work, many of which were not foreseen when the decision was made.

In summary, Panda demonstrates that application-unaware use of active networks is possible and can provide substantial benefits to applications. The automatic planning capability implicit in the idea can be realized with sufficiently low overhead and very high quality in the resulting plans.

References

[Fankhauser99] G. Fankhauser, M. Dasen, N. Weiler, B. Plattner, B. Stiller. "WaveVideo — An Integrated Approach to Adaptive Wireless Video." *Mobile Networks And Applications (Special Issue on Adaptive Mobile Networking and Computing)*, 4(4):255-271, December 1999.

[Feldmeier98] D. Feldmeier, A. McAuley, J. Smith, D. Bakin, W. Marcus, T. Raleigh. "Protocol Boosters." *IEEE Journal on Selected Areas in Communications (Special Issue on Protocol Architectures for 21st Century Applications)*, 16(3):437-444, April 1998.

[Fox97] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, P. Gauthier. "Extensible Cluster-Based Scaleable Network Services." *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP'97)*, Saint-Malo, France, October 1997.

[Fox98] A. Fox, I. Goldberg, S. Gribble, D. Lee, A. Polito, E. Brewer. "Experience With Top Gun Wingman: A Proxy-Based Graphical Web Browser for the USR PalmPilot." *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, Lake District, UK, September 1998.

[Gribble99] S. D. Gribble, M. Welsh, E. A. Brewer, and D. Culler. "The MultiSpace: an Evolutionary Platform for Infrastructural Services." *Proceedings of the 1999 USENIX Annual Technical Conference*, Monterey, California, June 1999.

[Hutchinson91] N. Hutchinson and L. Peterson, "The x-kernel: An Architecture for Implementing Network Protocols," *IEEE Transactions on Software Engineering*, vol. 17, no. 1, January 1991.

[Li02] J. Li, M. Yarvis, and P. Reiher, "Securing Distributed Adaptation," *Computer Networks*, Special Issue on Programmable Networks, vol. 38, no. 3, 2002.

[Mosberger96] D. Mosberger and Larry Peterson, "Making Paths Explicit in the Scout Operating System," *Proceedings of the Symposium on Operating Systems Design and Implementation*, October, 1996.

[Sudame98] P. Sudame, B. Badrinath. "Transformer Tunnels: A Framework for Providing Route-Specific Adaptaitons." *Proceedings of the USENIX Annual Technical Conference*, New Orleans, Louisiana, June 1998.

[Wetherall98] D. Wetherall, J. Guttag, and D. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," *Openarch 98*, 1998.

[Yarvis99] M. Yarvis, P. Reiher, G. Popek. "Conductor: A Framework for Distributed Adaptation." *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems (HotOS VII)*, Rio Rico, Arizona, March 1999.

[Yarvis00] M. Yarvis, P. Reiher, and G. Popek, “A Reliability Model for Distributed Adaptation,” *OpenArch 2000*, March 2000.