

Comma, A Communication Manager for Mobile Applications ^{*}

David Kidston J. P. Black Thomas Kunz [†] Michael E. Nidd [‡]
Marcello Liroy [§] Brent Elphick Michal Ostrowski

Shoshin Distributed System Research Group
Department of Computer Science
University of Waterloo

Abstract

Mobility is being touted as a next step in the computing revolution. Increased mobility implies increased reliance on wireless networks. However, techniques and tools for handling the variable and generally lower Quality of Service (QoS) offered by this type of network have not yet emerged. This problem is exacerbated by the fact that the standard communication protocols are unable to exploit such QoS as is available. Finally, advances in PCMCIA card technology mean that mobile applications must deal with a changeable computing platform. The combined inability of applications to adapt to a varying execution environment and the poor performance of communication protocols in a mobile environment have led us to develop a mobile application support architecture called the Communica-

tion Manager for Mobile Applications (Comma). Comma enables adaptive applications by providing methods for execution environment monitoring, protocol manipulation, and communication stream manipulation.

1 Introduction

Mobile applications execute in an environment that is characterized by a high degree of variability. Depending on a user's current location, the available bandwidth, error rates, and other QoS characteristics can change drastically. The modularity provided by plug-and-play PCMCIA cards means that the very capabilities of the portable device executing an application may also change. Power conservation and battery lifetimes are also important issues in a mobile environment.

For these reasons, researchers generally agree that mobile applications should be adaptive: depending on the characteristics of the execution environment, applications should change their visible behaviour and/or the way they utilize scarce resources. Comma was designed to support applications developed for these resource-

^{*}This work was done at the University of Waterloo with support from Motorola Canada Ltd. and the Natural Sciences and Engineering Research Council. Contact dkidston@uwaterloo.ca for more information.

[†]Now with the Department of Systems and Computer Engineering, Carleton University

[‡]Now at the IBM Zurich Lab

[§]Now at Qualcomm, Inc.

poor, dynamic environments. Comma provides access to a variety of computing and network statistics through a library and development environment (LDE) that connects applications to an execution environment monitor (EEM). The application is then notified of changes in these variables by either a callback to the application, or through periodic updates to a protected memory area. In this way, applications can alter their execution to best use the resources that are currently available.

Besides the inherent instability of the execution environment, mobile applications must deal with the legacy of wired network software, software which is now expected to work well in a completely different medium. Transport level protocols, such as TCP, were designed specifically for the error and delay characteristics of a wired medium. Inefficiencies arise when these protocols misinterpret the causes of problems on the wireless link. For example, when TCP encounters the poor QoS offered by wireless networks, it misinterprets the more frequent packet transmission errors as congestion and initiates congestion control and avoidance [3]. This reduces the throughput when the protocol should in fact be retransmitted immediately.

One solution to this problem would be to distribute a new set of protocols tuned for wireless links. Unfortunately, even if effective protocols were developed, it is unlikely they would be universally deployed. The technical and monetary overhead are not justified in what is still predominantly a wired world. Some research, including our own, has investigated the use of “backward compatible” protocols placed at a limited number of hosts (see, for example [1].) Such protocols operate transparently to the wired network, but improve the perceived QoS over the wireless link. This method confines both deployment

costs and computational overhead to the areas actually used by the mobile device. Comma supports protocol manipulation by intercepting and modifying packets bound to and from the wireless network using a service proxy (SP).

In this scheme, SPs are placed on the routers adjacent to the wireless portion of the network. Packets are intercepted at the router and then modified by packet filtering code. For example, packet streams could be dynamically compressed to use less bandwidth [6], prioritized by modifying TCP window sizes [9], or dropped altogether to save bandwidth in a layered encoding scheme such as MPEG. Thus, the SP can not only be used to modify protocols like TCP transparently, but can also be used by mobile applications to modify their communication streams.

This mechanism can also be used to make legacy applications appear adaptive. As long as the type and format of the communication between the client and server is known, packets bound to and from the mobile portion of the network can be modified or dropped in order to adapt to current network conditions. For example, consider an MPEG player and receiver. In this case, Comma could be used to drop the upper-level frames if available bandwidth were low. In this way, the stream itself is modified to be adaptive to network QoS, without changing either client or server.

In summary, Comma is designed to enable adaptive applications by providing methods for execution environment monitoring, protocol manipulation, and communication stream manipulation.

The remainder of the paper is structured as follows. Related research is covered in Section 2. Section 3 gives an overview of the Comma architecture. The current state of our implementation is given in Section 4 along with the results of ini-

tial testing. Security concerns associated with this architecture are covered in Section 5. Finally, a brief summary and ideas for future work are given in Section 6.

2 Related Work

In general, research in adaptive applications can be placed in one of four categories:

- network monitoring and event notification,
- specific adaptive mobile applications,
- toolkits for developing adaptive mobile applications, and
- transparent support of communication protocols, in particular TCP.

Research into network monitoring and event notification focuses on providing a means to detect changes in the underlying network and computation environment. It is also concerned with providing methods for notifying interested applications when such changes occur. While most papers differ in the exact way this information is collected and then passed on to interested applications, they all collect a similar minimal set of parameters such as available bandwidth, error rates, power status, etc. In [13], Welling and Badrinath describe a typical event-delivery framework for mobile applications. In this framework, an application registers interest in events of a given type, enabling it to track changes in the execution environment. Comma works similarly, providing access to a number of parameters through client registrations. However, the the set of parameters provided by Comma is configurable, so that besides SNMP

variables, user-designed sources of statistics can be monitored.

In order to better understand the needs of mobile applications, some specific applications have been designed and implemented. Two concrete examples in this second category are Fallmyr's adaptive diary [5] and Whalen's groupware editor [14]. In the former, the visible behaviour of the diary application changes depending on the available network bandwidth. For instance, long documents are not downloaded to the mobile host in low-bandwidth environments, and the user is presented with only a partial document. Whalen describes the design and implementation of a "mobile-aware" groupware editor. As the connection quality changes, the functionality changes as well, reflecting the richness or scarcity of bandwidth.

A number of researchers have described toolkits for the design and development of adaptive mobile applications. We briefly review three such toolkits. The Rover toolkit [7] combines relocatable dynamic objects (RDOs) and queued remote procedure call (QRPC) to provide unique services for "roving" mobile applications. An RDO is an object with a well-defined interface that can be loaded dynamically into a client from a server or vice versa. QRPC permits applications to continue making non-blocking RPCs even when a host is disconnected, and to use separate communication channels for request and response. MaROS [4] provides an object-oriented framework for client-server applications with support for dynamically changing mobile environments. The final version is claimed to support object migration between the mobile device and the MaROS server, a dedicated machine in the wired network. However, object migration is not yet supported, and migration decisions will have to be made by the application. Sync

[11] is a Java-based framework for collaborative mobile applications. Data objects are replicated between client(s) and a central server. The system emphasizes the detection and resolution of update conflicts, providing an application programmer with the ability to specify very data-type-specific synchronization operations.

All toolkits share a few common properties. They view a mobile application as a collection of objects that execute either on the mobile computer or on a server in the wired network. Adaptation to the current execution environment is achieved by delaying communication between objects and by executing objects on either the mobile host or in the fixed network. In all cases, it is left to the application programmer to specify explicitly how an application should adapt; the toolkits provide the mechanisms to implement the policies devised by the application programmer. Comma is intended to provide several types of support for application adaptation. A prototype mail reader that transparently makes object placement decisions based on information obtained from Comma is described in [10].

The last category of current research, perhaps closest to our own, aims to support mobile applications by improving the stability and performance of the transport-layer protocol over the unreliable wireless link. In particular, standard TCP is known to behave poorly over error-prone links, and papers such as [1, 2, 3] propose various mechanisms to improve TCP performance. Bakre and Badrinath [1] propose splitting the single TCP connection between a mobile host and a fixed host into two: one (standard) TCP connection between the fixed host and the last machine before the wireless link, and a second TCP connection between this last machine and the mobile host over the wireless link. The TCP implementation for the wireless link can adapt

to the characteristics of wireless links, improving the achievable throughput. However, splitting a single TCP connection into two violates the end-to-end semantics of the TCP protocol which could interfere with higher-level protocols. In [3], Balakrishnan *et al* propose a caching scheme for the wireless link, allowing the retransmission of lost packages without triggering flow control at the fixed host. In [2], they examine problems due to asymmetries frequently found in wireless networks, and their effects on the achievable TCP throughput. They propose a number of mechanisms to address these problems, some of which, however, require changes to the TCP protocol on the fixed host. Finally, Zenel [15] describes a general-purpose scheme to support various application-level protocols transparently using a proxy. A daemon running on a machine in the wired network sits in the communication path between the mobile and wired hosts. Protocol data units are processed by the proxy, which can, depending on the specific protocol and current execution environment, decide to filter certain packets, compress them, drop them, cache them, etc.

Comma provides a framework for implementing such enhancements to network-level protocols as well. However, we are currently concentrating on mechanisms that can be implemented without modifying the protocol stack at either the mobile or the wired host. In addition, we have identified a number of “value-added” TCP services, that not only allow us to improve the performance of TCP but also provide functionality otherwise not available in TCP (see [9]).

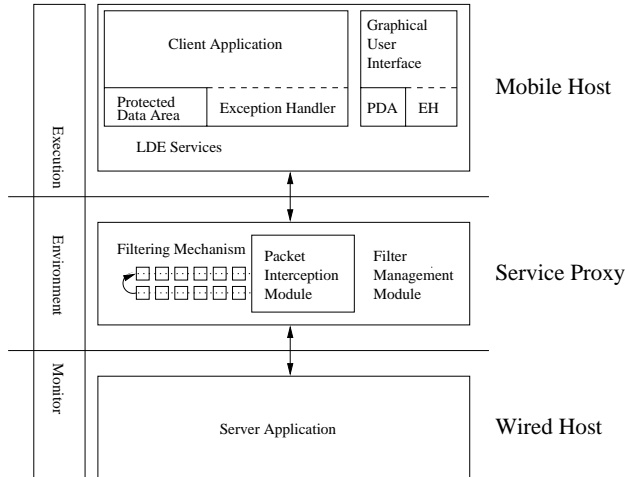


Figure 1: The Comma Architecture

3 The Comma Architecture

In order to support these adaptive mobile applications, we have designed a general architecture which includes:

- the Comma Execution Environment Monitor (EEM),
- the Comma Service Proxy (SP),
- the Comma Library & Development Environment (LDE), and
- the Comma Graphical User Interface (GUI).

EEM server daemons can run on any networked host, and gather local network and machine statistics. The EEM client runs as an application thread that communicates with the EEM server on hosts in which the application has registered an interest. The EEM server has been designed so that it can access a wide variety

of information sources on the local host. This allows application designers to extend the EEM to monitor application-specific variables.

Service Proxies (SPs) provide the ability to modify communication streams that travel to and from the mobile host. Packets are intercepted by the SP and passed to filters, which can alter the header and content of the packet. This allows applications to be partitioned, communication protocols to be transparently modified, and generalized services to be offered to packet-based communication.

The library and development environment (LDE) supports the creation of “mobile-aware” and adaptive applications by providing a single interface to the services of the SPs and of the EEMs. Application developers can use this environment to make applications reactive to changes in network and machine characteristics. It similarly allows configuration of EEM servers. The LDE also supports development of packet filters for SPs. Finally, it allows applications and network administrators to request the addition or removal of filters from communication streams.

The Comma Graphical User Interface (GUI) provides the user with a shell to the operation of the SPs and the EEM. The GUI has three main functions. Its primary role is a monitoring tool. The GUI enables direct observation of execution time statistics through its interface with the EEM. It also monitors the operation of an SP, indicating which streams are currently active, which filters are currently being applied to each stream and which filters are available. The GUI can also be used as a debugging tool by monitoring application interaction with execution measures and SP filters. Finally, the GUI is an interactive control tool. From the console, services for each packet stream can be requested

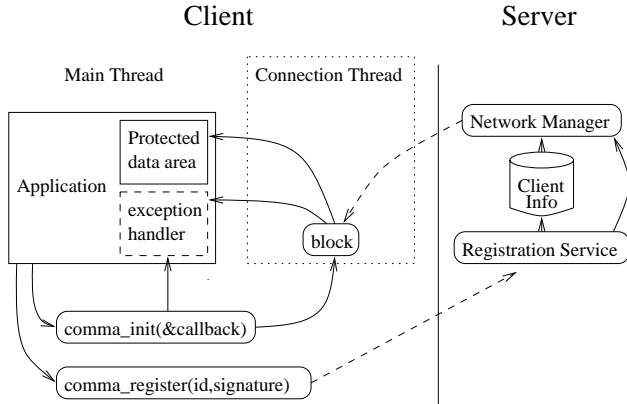


Figure 2: The Execution Environment Monitor (EEM) Architecture

or removed.

The following sections explore these components in more detail.

3.1 Execution Environment Monitor

The Comma Execution Environment Monitor (EEM) is a network and computing environment reporting tool. EEM servers run on suitable hosts and gather information on local performance metrics for local or remote clients. The EEM is configurable so that it can gather information from any local information source, including user-written ones.

The EEM design has four main parts: the client functional library, which presents an abstraction of the services offered by the EEM; the server process, which accepts and services requests from the clients; a client-supplied callback function, which is combined with an exception handler for interrupt style notifications; and a protected data area, which is used for periodic updates. The architecture is shown in Figure 2.

To use the EEM, clients, which may be applications or SP filters, call an initialization function specifying the address of a callback function if interrupt-style notification is desired. Initialization also clears the protected data area and starts a second thread to handle communication with EEM servers.

The client is now able to register an interest in network and execution environment metrics, or variables. The actual variables available at any EEM server will depend on the particular host, but it is expected that at least the SNMP variables will be available. It is hoped that, eventually, a standard set of metrics will be provided. However, the definition of a set of measures appropriate to all applications is beyond the scope of this paper.

To register interest in a variable, the client first creates a variable ID consisting of the variable name and the host on which the variable will be measured. This is accompanied by a “signature”, consisting of a range within which values of the variable most fall for notification to occur, and a method of notification.

Applications may be notified in one of two ways when an EEM finds that a registered variable falls within its requested range. The first is an interrupt-driven callback. If a notification arrives for a variable for which interrupt notification was requested, the exception handler immediately calls the callback function provided by the client on initialization. It is then up to the developer to handle the information passed to the function. The second and less intrusive method of notification involves periodic silent updates to a protected data store. The application can query the data store for whether a variable has changed, or what the most up-to-date value is.

Whenever a client registers for a variable on

an EEM server not already connected to the client, the connection thread opens a connection to the new host, sends the new variable registration information and then receive-blocks until it receives an update from the server. When information is received on this connection, the message is parsed by the exception handler and either a call to the callback function is made, or the common data area is updated.

The server initially waits for registrations from clients. Whenever it receives a request, it updates its database, taking note of the requesting host and port number. The server then makes periodic checks of the variables registered by all clients, and compares them to the conditions under which each client asked to be informed. If an interrupt-style variable has changed into the desired range, a notification message to the appropriate client is sent immediately. Otherwise, an update containing all variables that fall within their requested range is sent to the appropriate client once all variables have been checked.

This simple and extensible approach provides applications with the network and execution-environment metrics necessary for adaptation.

3.2 The Service Proxy

The SP provides a mechanism for manipulating, or filtering, packets bound to or from a mobile host. This single mechanism can be used to implement three different approaches to application partitioning. First, a filter can include part of the code for an application, resulting in application partitioning. Although not originally implemented for the purpose, this mechanism would be appropriate for dynamic object migration as shown by M-Mail [10]. Second, it can be used for transparent data-filtering purposes, such as web page compression [6] or DNS prefetching [12].

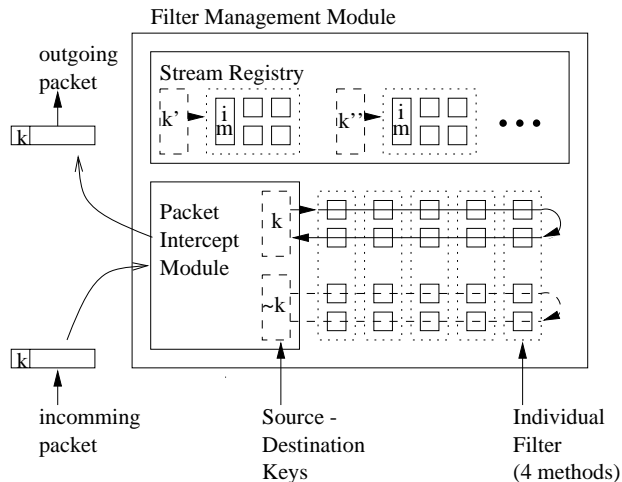


Figure 3: The Service Proxy (SP) Architecture

Third, the mechanism supports various types of protocol modification such as Snoop [3] or BSSP [9].

The SP design has four main components: packet interception, which removes packets from the network and matches the packet with a set of requested services; filter management, which assigns filters to new packet streams as well as handling dynamic addition and removal of filters from the filter pool; filter accounting, which keeps track of packet streams and the services applied to these streams; and, of course, the filters themselves. This architecture is shown in Figure 3.

In order to manipulate packets at the SP, we have designed a filtering method which takes a packet from the network, matches this packet with a set of filters, and then passes the packet to those filters for servicing. In order to uniquely identify communication streams, filters are associated with packet keys. A key consists of an

ordered quadruple that includes the source IP address and port, and the destination IP address and port. Together, these four uniquely identify a stream. Note that this implies that streams are directional. Most streams have an associated stream in the reverse direction which would have a key with the source and destination numbers reversed.

It is up to the application, or someone using the GUI, to specify which filters should be applied to which stream keys. In order to allow a filter to match multiple streams, portions of the key can be left blank, creating a “wildcard” key. A match is made if all but the blank portions of the wildcard key match the stream key. For instance, a wildcard key for a certain filter may give the destination IP address as the IP address of the mobile, and leave the rest blank. Then, all streams bound for any port on the mobile host will match.

Filter management keeps track of the filters currently available, and the keys associated with them. New filter–key bindings can be requested by the client, or by trusted third parties using the GUI. This process adds the key into the stream registry, and associates it with the desired filter and any parameters for the filter included in the registration. The filters themselves are kept in a filter pool, and can be compiled into the daemon as one of a standard set of services, or dynamically loaded during the operation of the daemon.

When a new packet reaches the SP, it is intercepted and presented to the packet-detection module for inspection. If the stream registry does not contain an entry for the exact key, this is the first packet of a new stream, and a “filter queue” for this stream must be created. A filter queue is conceptually a double queue of filter methods, an *in* and an *out* queue. The packet

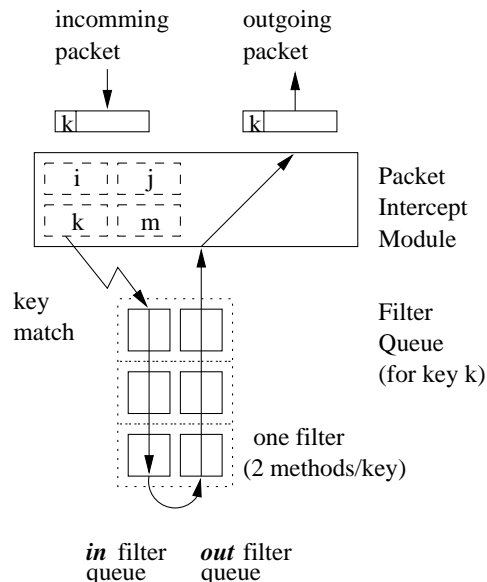


Figure 4: Detail of the SP filtering mechanism

is first passed to the first *in* method of the *in* queue, and then to the second and so on to the last *in* method. *in* methods are allowed only to read and not modify the packet. The packet is then passed to the last *out* filter method. This is the first method that can modify the packet. From there, the packet is passed to the second-last *out* method which can change the packet, potentially overwriting the modifications of the previous filter. The packet is then passed up the *out* queue until all filter methods have had their chance to modify the packet. If the packet has not been dropped completely, the resulting packet is reinjected into the network.

A filter queue is built by finding all filter objects in the stream registry whose associated wildcard key matches the packet key. Every filter has an insertion method associated with it that matches its other internal methods within either the *in* or *out* portion of a filter queue on a

specific key. Usually, the filter will use the key of the packet which caused the insertion method to be called, but it may add methods to other keys as well. It is quite common for the filter to add methods in the reverse direction of the stream, for example. Potentially, filters may add methods to completely unrelated streams. For example, if a filter wanted to monitor all the TCP streams of an HTTP proxy, it could insert methods on additional streams which were known to be part of the WWW session.

Once all methods for a key have been inserted by the various filter insertion methods, these methods are placed in order. The order is given by the absolute priority of a filter, decided when the filter is designed so that it does not interfere with the operation of other filters. For example, if a filter alters the content of packet, a higher-level filter may be required to alter the sequence numbers to make the change transparent. If the methods were applied in the opposite order, the packet would not be properly formed.

The full process, then, works as follows. When a new packet reaches the SP, it is intercepted and presented to the packet-detection module for inspection. If a filter queue already exists on the key, the packet is presented to the first *in* filter for that key. Once the packet has been read going down the filter queue, and modified on the way up, it is reinjected into the network. If a packet arrives at the SP for which no filter queue has yet been created, the packet-detection module searches its filter registry and attempts to match the packet key to the wildcard keys of registered filters. For each matching filter found, a new instantiation of the filter object is made, and the insertion method of that filter is called with the packet key. Once all methods have been added to specific stream keys, they are placed in a filter queue in priority order, and the packet

is passed through the queue as previously described.

Filter accounting is a side-effect of both packet detection and filter management. Whenever new streams are discovered and filters instantiated to service them, statistics are compiled internally. This information can be obtained using a special connection to the SP machine, and is currently used only by the GUI to display stream information to interested users.

3.2.1 Advanced Filter Operation

There are several more advanced operations a filter may want to do. For instance, if a filter is attempting to emulate half of a TCP stream, it may want to include time-outs and retransmissions. For this reason, a timer mechanism is included as part of the packet-interception module. Filter methods may send packets to the timer with a deadline, at which time the packet will be inserted into the appropriate filter queue as if it had arrived from the network.

3.2.2 SP placement

Due to the assumption that filters will see all packets bound to and from the mobile, SPs must be placed so that all network traffic passes through the router on which the SP resides. If some packets were to bypass the SP, the communication stream might arrive in an inconsistent state. The most logical solution is to integrate the placement of the SP with the Foreign Agents (FA) of Mobile IP. Since all packets bound for the mobile are routed to the FA via the Home Agent (HA), the FA is an obvious choice. As long as the FA is placed on the router into the subnet to which the wireless network antenna(s) are connected, all packets can be intercepted suc-

cessfully.

However, when a mobile passes from one wireless network to another, a hand-off mechanism is required. This issue has been partially dealt with in previous prototypes of Comma, but currently no mechanism exists. See the future work section for more on this issue.

3.3 Library and Development Environment

To aid in the creation of mobile-aware and adaptive applications, Comma provides a programming interface to SP and EEM functionality. The library allows applications to register an interest in an execution variable, ask for a service to be provided to a communication stream, or ask for a service to be discontinued. It also supports the design of new filters and EEM information sources.

As described in Section 3.1, if an application wishes to be notified when a variable is within a certain range, the LDE provides a set of functions that allow the application to communicate with the appropriate EEM.

If an application wishes some of its packet streams to be serviced by a set of filters, the application registers the filter ID and detection key with the current SP. The filter management module of the SP will then be contacted to begin associating the stream keys with the requested filters. Services can be removed using a similar method.

The LDE can also be used to develop new processor and network metrics to be added to the set of variables offered by an EEM server. By using the interfaces specified, the server contacts the user-defined source and sends the reported values to any client that requests it from the EEM server.

Finally, the LDE provides the libraries required for the creation of new SP filters. Combined with the functionality supporting EEM, this can be used to create a partition of an application to run on the wired network while the rest of the application runs on the wireless machine. The wide range of possible uses for filters has been discussed in the Section 3.2.

3.4 GUI

The Graphical User Interface provides a shell into the operation and services offered by Comma. Its main purpose is to monitor the environment in which mobile applications are running. First, it keeps track of communication streams passing through individual SPs. By connecting with an individual SP, the GUI can provide information such as what streams are currently passing through the SP, the services offered to each stream, and the current status of each stream. It can also tell what filters are available at this SP and the wildcard keys that have been associated with each filter. Eventually it is hoped that automatic stream-application matching will be available. Second, it monitors the variables on mobile and fixed hosts. The GUI connects with EEM servers and displays any desired measure. Care should be taken if using the GUI on a mobile device as it may exacerbate wireless network congestion.

The GUI can also be used as an intervention tool. From the GUI, users can associate and dissociate filters with keys. This is especially important for legacy applications. Since these applications can't request services on their own behalf, it is up to the user to request services using the GUI. Protocol-enhancing services can be added. Also, if the type and format of the communication used by the application is known,

application-specific protocols that make use of this knowledge can be created (using the LDE). Then, graceful degradation or lossy compression can be used intelligently in the face of low or varying QoS on the wireless network.

Finally, the GUI can be used as an debugging tool. Since the user is able to monitor things like the addition and removal of filters by applications, developers can make sure that applications are following their specifications. The developer can also use the GUI to monitor the machine and network utilization of their code.

4 Implementation and Testing

We have implemented a prototype of the Comma architecture, and have completed a number of preliminary experiments.

The EEM is now fully functional and is in the process of being documented. It is being used by the GUI to give information on local network statistics, and for testing purposes. The EEM server uses SNMP as its main data source, but several other variables are offered. These variables were found to be of use for earlier application development at our lab (see Table 1).

The SP has been through two development cycles. At the moment, it provides a limited number of simple services, including the TCP prioritization mechanism proposed by Liroy [9], and a packet-dropping filter which will be integrated with an MPEG server and player to reduce bandwidth on the wireless link when necessary. The packet-dropping and future compression filters rely on a TCP-level filter that maintains TCP end-to-end semantics by adjusting packet and acknowledgment sequence numbers to compensate for the corresponding changes to the data stream. Results from these and other services

<i>variable</i>	<i>description</i>
netLatency	measure of the network latency using ping RTTs
avgInIPPkts	average of incoming IP packets uni- or broadcast
cpuLoadAvg	cpuload average, as recorded by the kernel
ethErrsAvg	number of errors in ethernet frames received by host
ethInAvg	number of incoming ethernet frames received by host
ethOutAvg	number of outgoing ethernet frames sent by host
deviceList	string that lists the devices configured on host
bytes_rx	bytes received by the network device driver
bytes_tx	bytes transmitted by the network device driver

Table 1: Additional EEM Variables

will be explored in future papers.

The LDE has been integrated with the EEM, and is partially integrated with the SP. Similarly, the GUI is integrated with the EEM, but not the SP.

5 Security Issues

The Comma architecture provides unique opportunities for developing adaptive applications, but also has some associated risks. Since communication streams are intercepted transparently, there is the risk of eavesdropping, or impersonation. Making the environment in which Comma operates secure should be of great concern.

This design is intended to be integrated into router implementations at or near wireless net-

works, and to provide additional services for mobile users. The question becomes one of trust towards those that administer the routers. As long as the router is itself secure, Comma does not represent any more of a security risk than what could already be done at the router. It is thus up to those who provide Comma services to take accountability for its use. The very power of Comma makes general availability dangerous. However, properly used, it provides a powerful aid in the design, implementation and use of mobile, adaptive applications.

6 Summary and Future Work

In summary, Comma combines a number of ideas found in the literature, integrates them into a common framework, and allows for the provisioning of value-added transport-layer services, execution environment monitoring, and communication stream manipulation. Comma gives applications access to network statistics, allowing them to adapt their execution to the available QoS. Finally, Comma provides a monitoring and design environment to simplify the task of developing new and innovative adaptive mobile applications.

Much remains to be implemented to reach the full capabilities of the architecture. One problem to be solved in the near future will be the “hand-off” of a mobile user from one SP to another. Once a mobile moves from one wireless network to another, state information will have to be transferred between SPs. In order to maintain seamless and transparent service, all filter state must be sent from one Proxy to another. One proposal is to use a method similar to Java applets [8]. They transfer entire objects in mid execution from host to host. There are, how-

ever, severe performance and security issues associated with them. Some design work has been done in previous prototypes of Comma, where peer SP objects send each other state information, but no final design decision has been made.

Another limitation of this current implementation of Comma is that it only processes TCP packets. All other packets are passed back to the network without inspection. Eventually it is hoped that other transport and even network-level filters will be developed.

References

- [1] Ajay Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 136–143, Vancouver, BC, Canada, May 1995.
- [2] Hari Balakrishnan, Venkata N. Padmanabhan, and Randy H. Katz. The effects of asymmetry on TCP performance. In *Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 77–89, Budapest, Hungary, September 1997.
- [3] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy H. Katz. Improving TCP/IP performance over wireless networks. In *Proceedings of the First Annual International Conference on Mobile Computing and Networking*, pages 2–11, Berkeley, California, USA, November 1995.
- [4] Sebnem Baydere and MaROS Group. Maros: A framework for application development on mobile hosts. In *Proceedings of the IASTED International Conference*

- on *Parallel and Distributed Systems (Euro-PDS '97)*, pages 269–274, Barcelona, Spain, June 1997.
- [5] Terje Fallmyr. Adaptable mobile systems. In *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 363–368, Cheju Island, Republic of Korea, August 1995.
- [6] A. Fox, S.D. Gribble, E.A. Brewer, and E. Amir. Adapting to network and client variability via on-demand dynamic distillation. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 160–170, Cambridge, MA, October 1996.
- [7] Anthony D. Joseph, Alan F. deLespinasse, Joshua A. Tauber, David K. Gifford, and M. Frans Kaashoek. Rover: A toolkit for mobile information access. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 156–171, Copper Mountain Resort, Colorado, USA, December 1995. Appeared as *ACM Operating Systems Review*, 29(5), December 1995.
- [8] Danny B. Lange and Daniel T. Chang. *IBM Aglets Workbench, Programming Mobile Agents in Java, A White Paper*, September 1996.
- [9] Marcello Lioy. Providing TCP-level services to mobile computers in wireless networking environments. Master's thesis, Dept. of Computer Science, University of Waterloo, August 1997.
- [10] Hai Yan Lo. M-mail: A case study of dynamic application partitioning in mobile computing. Master's thesis, Dept. of Computer Science, University of Waterloo, May 1997.
- [11] Jonathan P. Munson and Prasun Dewan. Sync: A Java framework for mobile collaborative applications. *IEEE Computer*, 30(6):59–66, June 1997.
- [12] Stuart Wachsberg. Efficient information access for wireless computers. Master's thesis, Dept. of Computer Science, University of Waterloo, September 1996.
- [13] Girish Welling and B. R. Badrinath. A framework for environment aware mobile applications. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, pages 384–391, Baltimore, Maryland, USA, May 1997.
- [14] Tara J. Whalen. Design issues for an adaptive mobile group editor. Master's thesis, Dept. of Computer Science, University of Waterloo, September 1997.
- [15] Bruce Zenel and Dan Duchamp. A general purpose proxy filtering mechanism applied to the mobile environment. In *Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 248–259, Budapest, Hungary, September 1997.

David Kidston is a Masters candidate at the University of Waterloo. He received his B.Ed and Honours B.Sc (Computer Science) from Queen's University in 1996 and 1995 respectively. Now working on his thesis, his current research interests include QoS and security issues in wireless networks, network management and the various improvements to IP, including Mobile IP, and RSVP. He is a member of the Mobile Project within the larger Shoshin research group at the University of Waterloo.

J.P. Black received his Ph.D. in Computer Science from the University of Waterloo in 1982, having previously studied at the University of Calgary and the Institute National Polytechnique de Grenoble, France. He has been on the faculty of the University since 1984. He is currently Associate Provost, Information Systems and Technology, and is an Associate Professor in the Department of Computer Science. His current research interests include the management of distributed applications and systems, distributed debugging and the visualization of complex executions, and mobile and wireless computing. His research forms part of the activities of the Shoshin research group at Waterloo.

Thomas Kunz received the Dr. Ing. degree from the Technical University of Darmstadt, Federal Republic of Germany, in May 1994. From 1994-1997 he worked as assistant professor in the Department of Computer Science at the University of Waterloo, Ontario, Canada, where the research reported in this paper was conducted. He recently joined the Department of Systems and Computer Engineering at Carleton University in Ottawa, Ontario, Canada, as assistant professor. His research in-

terests include load balancing in distributed systems, distributed debugging, management of distributed applications and systems, mobile computing, and reverse engineering/program understanding.

Michael Nidd received his B.Math and M.Math degrees from the University of Waterloo prior to working as a research associate on the Shoshin Project. He is currently working at the IBM Zurich Laboratory.

Marcello Liroy recently received his M.Math degree at the University of Waterloo, and is currently employed by Qualcomm in San Diego.

Brent Elphick and **Michal Ostrowski** have worked on the Shoshin project as undergraduate research programmers. Brent Elphick received his B.Math at University of Waterloo in 1998.