Protocol Boosters

D. C. Feldmeier, A. J. McAuley, J. M. Smith
D. S. Bakin, W. S. Marcus, T. M. Raleigh

dcf@music.com, mcauley@bellcore.com, jms@cis.upenn.edu
dbakin@bellcore.com, wsm@bellcore.com, tom@bellcore.com

Bellcore and University of Pennsylvania*

Abstract

This paper describes a new methodology for protocol design, using incremental construction of the protocol from elements called "protocol boosters" on an as-needed basis. Protocol boosters allow: (1) dynamic protocol customization to heterogeneous environments, and (2) rapid protocol evolution.

Unlike alternative adaptation approaches, such as link layer, conversion, and termination protocols, protocol boosters are both robust (end-to-end protocol messages are not modified) and maximize efficiency (does not replicate the functionality of the end-to-end protocol). We give examples of error and congestion control boosters and give initial results from booster implementations.

1 Introduction and Problem Statement

At the heart of the success and power of the worldwide IP Internet are the general purpose protocols in the TCP/IP protocol suite. Although these protocols, such as TCP and IP, provide the flexible framework for building diverse applications, two limitations can be seen precisely because of their success and generality:

- They evolve slower than the changes in networking technology and application requirements.
- They trade some loss in efficiency for their ability to handle increased heterogeneity.

1•1 Slow Evolution of Protocols

Network technology and applications are changing rapidly, and existing protocols may not operate well in new circumstances. For example, the fast growth of the Internet has raised the potential problem of address exhaustion in IP version 4, leading to the creation of IP version 6. Also, with the rise of real-time and multicast applications (using UDP) and the rise of applications that frequently open and close TCP connections (notably HTTP [13]), new congestion control mechanisms may be needed to maintain network efficiency and reliability.

* This work was supported by the Defense Advanced Research Projects Agency under contract #DABT63-95-C-0073. Additional Support at the University of Pennsylvania was provided by the AT&T Foundation, the Hewlett-Packard Corporation and the Intel Corporation.

Although the rate of change of technology and applications is increasing, the rate of change of the key Internet protocol standards has slowed. IP version 6, for example, took many years to standardize and still is not widely adopted. Paradoxically, it is the success of the protocol that leads to this problem. The rate of change becomes political (defined by standards bodies) rather than technological. The problem is not with the specific protocol or specific standardization method; but with the need to have many people (sometimes with competing agendas) agree on the standard. Forming a consensus within large groups is a slow process, and is likely to remain slow; therefore, protocol standards will continue to evolve at a slow pace.

1•2 Inefficiency of General Purpose Protocols

General purpose protocols are designed to operate in heterogeneous network environments by minimizing the services required from lower layers. Minimizing lower layer service requirement allows robust operation over the widest variety of network infrastructures, but prevents the protocol from taking full advantage of lower layer services.

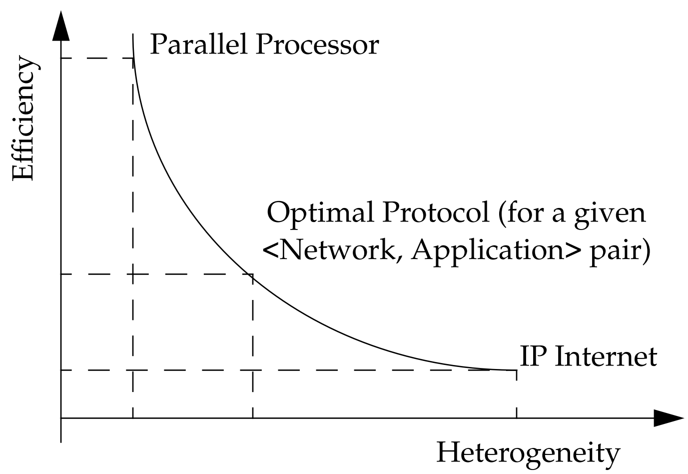

Figure 1   Trade-off between efficiency and heterogeneity

Figure 1 shows an abstract example of the efficiency-generality trade-off: with IP Internet protocols being general purpose but inefficient and parallel processing protocols being efficient but requiring low error rate, low delay networks. Ideally, a protocol would adapt to provide the best possible performance given the path of the data. For instance, when on a LAN, the protocol would adjust itself to give performance similar to that of a specialized LAN protocol.

1•3 Summary

As the political pace at which standards evolve is unlikely to increase and the heterogeneity of the network is likely to remain high, we need a new means of creating efficient protocols that can:

- Evolve rapidly.
- Dynamically change behavior and functionality based on the visible network heterogeneity and application requirements, rather than the total network and application heterogeneity.

Section 2 describes a new design methodology for protocols called protocol boosters, and Section 3 explains how protocol boosters allow faster protocol evolution and optimization of the protocol to the environment. Section 4 compares protocol boosters with other protocol architectural alternatives. Section 5 provides some initial implementation and experiment results.

2 Protocol Boosters

This section defines protocol boosters and shows how a protocol, combined with a set of protocol boosters, forms a family of protocols. We also describe examples of protocol boosters.

2•1 What is a Protocol Booster?

A protocol booster is a software or hardware module that transparently improves protocol performance. The booster can reside anywhere in the network or end systems, and may operate independently (one-element booster), or in cooperation with other protocol boosters (multi-element booster). Protocol boosters provide an architectural alternative to existing protocol adaptation techniques, such as protocol conversion and protocol termination.

A protocol booster is a supporting agent that by itself is not a protocol. It may add, delete or delay protocol messages, but never originates, terminates, or converts that protocol. A multi-element protocol booster may define new protocol messages to exchange among themselves, but these protocols are originated and terminated by protocol booster elements and are not visible or meaningful external to the booster. Figure 2 shows the information flow in a two-element booster.

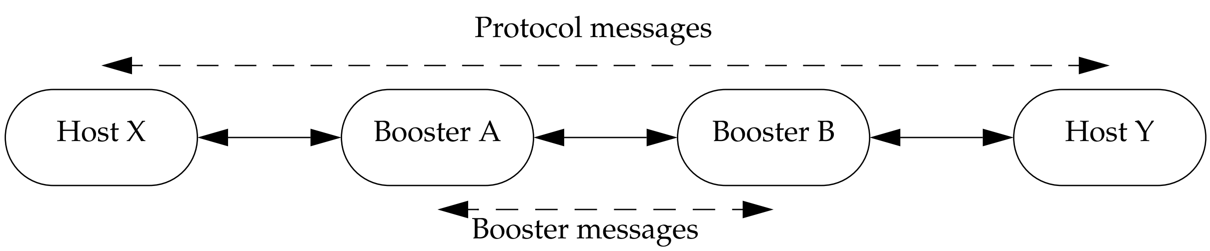

Figure 2 Two-element booster

A protocol booster is transparent to the protocol being boosted. Thus, the elimination of a protocol booster will not prevent end-to-end communication, as would, for example, the removal of one end of a conversion (e.g., TCP/IP header compression unit [10]) or termination protocol. A simple analogy may help explain the boosting and transparent behavior. If a protocol is analogous to an automobile tire, then a protocol booster is analogous to snow chains. A car with regular tires can add snow chains on snowy roads, just as a protocol designed for a wireline network may use boosters for transmission across a wireless network. The snow chains and protocol boosters are supporting agents since a car cannot drive on snow chains alone, just as communication is not possible with protocol boosters alone. Also, the snow chains and boosters are transparent, since the car tires are not modified by the addition of snow chains, just as the protocol is not terminated or converted by a protocol booster.

2•2 Protocol Family

Multiple protocol boosters can operate simultaneously on the same protocol. Although a booster itself is not a protocol, a protocol and a booster combine to form a new protocol. The new protocol consists of the messages of the original protocol combined with any messages exchanged among multi-element booster elements. We can implement a second booster for the new protocol by concatenating or nesting it within the booster elements of the first booster. Booster concatenation or nesting can continue to an arbitrary depth.

With a set of boosters, it is natural to talk about the protocol family generated by the boosters. The family always has a "base protocol" that is operated upon by all the booster elements. Additional protocol family members are generated by adding booster elements. Thus, protocol boosters can be the modular building blocks for a family of related protocols, each suited to a specific environment. Ideally, we design a protocol family from scratch, starting with the minimum protocol and adding booster layers as options. Protocol boosters are, however, just as useful when applied to existing protocols.

2•3 One-Element Protocol Booster Examples

This section gives examples of one-element boosters. As there is only one booster (Figure 2 without the booster B), there are no new protocol messages defined.

2•3•1 One-Element Error Detection Booster for UDP

UDP has an optional 16 bit checksum field in the header. If it contains the value zero, it means the checksum was not computed by the source. Computing this checksum may be wasteful on a reliable LAN; however, if errors are possible, the checksum greatly improves data integrity. A transmitter sending data does not compute a checksum for either local or remote destinations. For reliable local communication this saves the checksum computation (at the source and destination). For wide-area communication, the single-element error detection booster computes the checksum and put it into the UDP header. The booster could be located either in the source host (below the level of UDP) or in a gateway machine.

2•3•2 One-Element ACK Compression Booster for TCP

On a system with asymmetric channel speeds, such as broadcast satellite, the forward (data) channel may be considerably faster than the return (ACK) channel. On such a system, many TCP ACKs may build up in a queue, increasing round trip time and thus reducing the transmission rate for a given TCP window size. The nature of TCP's cumulative ACKs means that any ACK acknowledges at least as many bytes of data as any earlier ACK. Consequently, if several ACKs are in a queue, it is necessary to keep only the ACK that has arrived most recently. A simple ACK compression booster could assure that only a single ACK exists in the queue for each TCP connection. (A more sophisticated ACK compression booster allows some duplicate ACKs to pass, allowing the TCP transmitter to get a better picture of network congestion.) The booster increases the protocol performance because it reduces the ACK latency, and allows faster transmission for a given window size.

2•3•3 One-Element Congestion Control Booster for TCP

Congestion control reduces buffer overflow loss by reducing transmission rate at the source when the network is congested. A TCP transmitter deduces information about network congestion by examining acknowledgments (ACKs) sent by the TCP receiver. If the transmitter sees several ACKs with the same sequence number, then it assumes that network congestion caused a loss of data messages. If congestion is noted in a subnet, then a congestion control booster could artificially produce duplicate ACKs. The TCP receiver would think that data messages have been lost because of congestion and would reduce its window size; thus, reducing the amount of data it injects into the network.

2•3•4 One-Element ARQ Booster for TCP

TCP uses ARQ to retransmit data unacknowledged by the receiver when a packet loss is suspected, such as after a retransmission time-out expires. If we assume the network of Figure 2 (except that booster B does not exist), then an ARQ booster for TCP will: a) cache packets from Host Y, b) if it sees a duplicate acknowledgment arrive from Host X and it has the next packet in the cache, then it deletes the acknowledgment and retransmits the next packet (because packet must have been lost between the booster and Host X), and c) delete packets retransmitted from Host Y that have been acknowledged by Host X. The ARQ booster improves performance by shortening the retransmission path. A typical application would be if Host X were on a wireless network and the booster was on the interface between the wireless and wireline networks. This ARQ booster is a simplification of the Snoop protocol developed at Berkeley [4].

2•4 Two-Element Protocol Booster Examples

Two-element boosters appear similar to independent link protocols. However, some of the examples contained in this section show that protocol boosters can be more efficient than equivalent link protocols. A more general discussion of the advantages of protocol boosters as compared with link protocols is contained in Section 4.1.

2•4•1 A Forward Erasure Correction Booster for IP or TCP

For many real-time and multicast applications, Forward Error Correction coding is desirable [12]. The two-element FZC booster uses a packet Forward Error Correction code and erasure decoding [2]. The FZC booster at the transmitter side of the network adds parity packets. The FZC Booster at the receiver side removes the parity packets and regenerates missing data packets. The FZC booster can be applied between any two points in a network (including the end systems). If applied to IP, then a sequence number booster adds sequence number information to the data packets before the first FZC booster. If applied to TCP (or any protocol with sequence number information), then the FZC booster can be more efficient because a) it does not need to add sequence numbers and b) it could add new parity information on TCP retransmissions (rather than repeating the same parities). At the receiver side the FZC booster could combine information from multiple TCP retransmissions for FZC decoding.

2•4•2 Two-Element Jitter Control Booster for IP

For real-time communication, we may be interested in bounding the amount of jitter that occurs in the network. A jitter control booster can be used to reduce jitter at the expense of increased latency. At the first booster element, time-stamps are generated for each data message that passes. These time-stamps are transmitted to the second booster element, which delays messages and attempts to reproduce the inter-message interval that was measured by the first booster element.

2•4•3 Two Element Selective ARQ Booster for IP or TCP

For links with significant error rate using a selective ARQ protocol (with selective acknowledgment and selective retransmission) can significantly improve efficiency compared to using TCP's ARQ (with cumulative acknowledgment and possibly go-back-N retransmission). The two-element ARQ booster uses a selective ARQ booster to supplement TCP by: a) caching packets in the upstream booster, b) sending negative acknowledgments when gaps are detected in the downstream booster, c) selectively retransmitting the packets requested in the negative acknowledgments (if they are in the cache).

2•5 Booster Policy

Booster policy says when, and how much of, a booster's function is invoked. For example, a FZC booster might be used over a wireless data link to bring its error behavior into an application's particular acceptable operating range. The policy function takes the form of a control algorithm that determines the amount of overcode necessary given the observed state of the wireless network.

Boosters can be added and deleted dynamically as additional network functionality is needed. A policy for this decision is needed in addition to the specific booster mechanism for adding functionality. Since boosters vary widely in their functions, it is impossible to have a completely general policy; policies must be associated with specific booster functionality. Policies may be based on a wide variety of factors, such as observed network behavior, packet source and destination, or time of day.

Booster policies are needed to dictate the ordering of some boosters. A policy module can be devised that structures the interaction of boosters, for example by indicating that two boosters are not commutative. Consider two boosters, an FZC booster and the ACK compression booster. If FZC is applied first, then the number of redundant packets is based on the number of ACK packets sent by the receiver. However, if the ACK compression booster is applied first, then the number of redundant packets is based on the compressed number of ACK packets. Thus fewer ACK packets are transmitted if FZC is applied second.

3 Protocol Boosters as a Solution to Current Networking Problems

In this section, we discuss how protocol boosters overcome the slow evolution and reduced efficiency of standard general purpose protocols.

3 • 1 Fast Evolution of Protocol Boosters

With no need for standardization, we can design and implement protocol boosters with minimal resources. Also, because boosters are transparent, we can replace boosters as often as desired without the knowledge of those using them. Thus, a simple, quick booster implementation can be installed quickly. As experience is gained, improved boosters can be created and installed. Because of the fast feedback that can be obtained on booster behavior and performance, boosters can evolve extremely quickly. Also, many different boosters can evolve independently in parallel. All of this is because the transparent nature of boosters eliminates the need for standardization.

Protocol boosters are a free-market approach to protocol and network design. Booster designs compete economically in the marketplace, rather than politically in a standards committee. In economics, it is generally believed that companies that compete economically in a free-market system are more efficient than those companies that operate by government fiat. Companies that are successful in the market efficiently produce goods and services desired by consumers. Similarly, the free-market competition among boosters assures that efficient and effective boosters will proliferate and that poor booster designs will become obsolete. Standardization of protocols is expensive because of the need to attend standards meetings, and only established companies can afford to be involved in standards. Boosters need not be standardized, and they can be quickly designed and built by a small number of people at low cost.

Protocols that require standardization are subject to what economists call "network externality". Network externality is the concept that the value of something depends on the number of people who already use it. Examples of network externality can be seen in VCRs (Video Cassette Recorders) and computer operating systems. VHS tapes are the most available because most people have VHS VCRs. Once one specific example of something that fills a niche becomes dominant, it is difficult to displace it even with a technically superior product. Standardized protocols are subject to network externality. We use TCP/IP because we want to communicate with other people, and most of them use TCP/IP. Network externality dampens competition, because even if a better protocol is designed, it is unlikely to displace existing protocols. Boosters are immune from the feedback caused by network externality because they are transparent and need not be standardized. Consequently, existing boosters can and will be displaced easily and quickly by boosters with better performance.

Another advantage of boosters is that their design and use can be proprietary. With standardized protocols, proprietary market advantage is not possible because you can only communicate with those systems that are running the standardized protocol. Boosters are transparent, and thus, there is no need to disclose the internal design of a booster to those using the booster. The ability to gain proprietary advantage using booster means that there is increased market incentive to invest in new booster designs. Care must be taken, however, that the proliferation of protocol boosters does not result in poor performance because of unexpected interactions among proprietary boosters. To reduce this lack of interoperability, successful proprietary booster protocols could eventually become standardized, at which time the developer gives up proprietary claims in return for the wider market for standardized solutions

3•2 Efficiency of Protocol Boosters in Heterogeneous Networks

It is difficult to maintain efficient protocol operation over a wide range of network environments. Thus, as shown in Figure 1, protocols generally exchange efficiency for heterogeneity. Protocol efficiency often can be increased by reducing the heterogeneity in a networking environment. Boosters can be used to increase protocol efficiency without reducing heterogeneity because boosters are a means of hiding the heterogeneity of the networking environment. Instead of optimizing the performance of a protocol over a wide range of network environments, a protocol can be optimized for the network environment between the end host and a booster or between boosters. For example, a protocol designed to operate over local area networks (LANs) can have high performance compared with a more general protocol because LANs present a narrower range of operating conditions than a general internet environment. Protocol boosters allow the use of the simplest protocol of a protocol family for a given application and the network over which the data is carried. Because boosters can be added or removed easily, it is simple to remove unneeded functionality. For example, if network congestion is not an issue on a local network, the booster that adds congestion control can be disabled to reduce unnecessary delay. Thus, boosters allow dynamic customization of a protocol to a heterogeneous environment. It is not necessary to make pessimistic worst-case assumptions about network conditions and application requirements, as is necessary for general purpose protocols.

The network environment seen on an end-to-end communication path depends on the route taken by the data through the communication network. When boosters are placed in the network, boosters are knowledgeable about the network environment in which they operate. Network boosters also operate only on data that passes through them, so data is only processed at the level necessary for communication locally. Any non-local communication is automatically enhanced by boosters. Thus, boosters provide the highest possible performance given the route of the data.

Boosters can improve the performance of protocols that have a “fast-path” [5] implementation that optimizes performance when things are going well (e.g., packets arrive uncorrupted and in the same order they were sent). For example, when a booster improves the performance seen across a noisy wireless links, TCP can operate mostly using the fast-path; without the booster, fast-path use would be rare.

4 Comparison of Boosters with Other Approaches

Protocol boosters provide faster evolution and increased efficiency compared to the use of standard general purpose end-to-end protocols. This section compares protocol boosters with other protocol architecture alternatives, noting that only boosters take advantage of higher layer information (unlike link layer adaptation), while not altering the message syntax (protocol conversion) or semantics (protocol termination).

4•1 Link Layer Adaptation

Link layer protocols sometimes perform higher layer functions, such as error control on noisy links and encryption on insecure links. As with booster protocols, link protocols can be finely tuned to the link's characteristics. The main distinction is that link protocols are independent of high layer protocols. This independence (layering) has advantages and disadvantages.

Layering isolates a protocol from changes in other layers. In contrast, when a base protocol changes, the protocol booster may need to change. Also, if a base protocol is encrypted, then the protocol booster must use another base protocol (e.g., with IPSEC encryption, the FZC booster must go from booster TCP to boosting IP). Although modularity is an often stated benefit of layering, a universal framing structure [7] can maintain modularity without layering †.

Violating artificial layer boundaries allows higher performance. Implementations using "Integrated Layer Processing" [6], for example, avoid unnecessary copying. Violating layering also avoids other duplicated functionality. For example, if the transport layer protocol provides error control (e.g., TCP), then link error control may do unnecessary retransmissions (and if the transport layer does not provide error control (e.g., UDP), then end-to-end robustness is lost [14]).

Violating layering allows more selectivity. If applications with divergent needs pass over the same link, as they frequently do, then it is unlikely that a single link layer protocol can provide the most efficient service. When data is sent over a noisy link, for example, some data streams, such as those carrying audio, may desire low delay, even if some errors occur. Other data streams may desire a low residual error rate, even at the expense of increased delay. No single link layer protocol can satisfy both needs. In contrast, a protocol booster could ignore all UDP connections or boost only specific applications (as determined by TCP/UDP port numbers). Moreover, the booster could be migrated into the end system where it more under the control of the application.

4•2 Protocol Conversion

Protocol conversion [8] converts from one protocol to another, while maintaining the semantics of the original protocol. Van Jacobson's TCP header compression, for example, converts TPC/IP headers into a compressed syntax. The compression increases the throughput of TCP over slow network links by taking advantage of the fact that many fields in the TCP/IP headers rarely change. The main distinction between conversion protocols and booster protocols is that conversion changes the syntax of the base protocol.

A disadvantage of protocol conversion is that it requires processing to change message syntax. While a protocol booster simply can observe most messages from the base protocol, the protocol converter must modify every message that arrives. Another disadvantage of protocol conversion is that it not robust to failures. While protocol boosters and link protocols offer soft degradation, protocol conversion and protocol termination offer hard degradation.

---

† Chunks [7] separates protocol syntax and semantics so that no violation of modularity is required for proper booster operation.

4•3 Protocol Termination

Protocol termination uses different protocols at different points along the path from the transmitter to the receiver, with no single end-to-end protocol. Just as with protocol boosters, protocol termination allows choice of protocol appropriate to the network environments along the communication path and avoids duplication in functionality. For example, a TCP connection from a fixed host could be terminated at the border of a wireless network, and a more efficient error control protocol could send the packets over the wireless network [3] [1].

A drawback of protocol termination is that it loses desirable end-to-end properties. For example, if TCP is terminated in the network, then receiving a TCP acknowledgment does not mean that information has arrived at the destination. Another drawback of protocol termination is that it provides additional points of failure. Failure of a network termination point causes all messages to be unintelligible (hard degradation), even if alternative routes are available. If a booster fails, communication is still possible using the base protocol as long as another communication route is available. Performance degrades, but protocol operation continues (soft degradation). Soft degradation is helpful in hostile environments in which failures are expected, such as battlefield situations.

4•4 Special Purpose End-to-End Protocols

Instead of using a standard general purpose end-to-end protocol, applications can use protocols adapted to their needs and the network being used. Several different adaptation techniques have been discussed in the literature.

Applications may implement the majority of protocol functions, including acknowledgment, retransmission, and forward error correction, as required. To increase reuse and avoid application programmers needing to become protocol experts, work has been done to aid in composing protocols from basic functions, for example Arizona's x-Kernel [9]. Protocol boosters also can be used for protocol adaptation in the endpoints; however, the flexibility of allowing the functionality anywhere in the network has a number of advantages.

Although it is usually preferred to do protocol processing in the endpoints [14] sometimes it can be more efficient to do the equivalent processing in the network. For example, video compression may be better done in the network. Usually, the local network is sufficiently fast to handle uncompressed video. Video compression is required for wide area networks with lower bandwidths. Rather than provide a hardware accelerator for each workstation that requires video compression, we can use a single video compression booster at the edge of the local area network, where the need for compression is delineated.

Putting boosters in the network allows greater and more up-to-date knowledge of network behavior and state. Somehow the end point adaptor must map destination address to route to network characteristics along a route. In some cases, the mapping is quite simple. For example, if the source and destination share the same network address, then they are on the same network, and presumably, the characteristics of the local network are known. In general, however, building and maintaining such a database is hard. Moreover, with the expected increase in mobile clients and Network Address Translators the task will become harder.

An incorrectly designed booster could disrupt end-to-end robustness properties. For example, it may systematically discard protocol messages in such a way as to deadlock the protocol. However, it is possible to design boosters to prevent this problem. A simple method is to have the booster monitor protocol progress. If progress is not satisfactory, then the booster can stop or reinitialize its state. Performance degrades, but protocol operation continues.

There are some situations where placing functionality in the network is not desirable or possible. If an IP packet is encrypted, for example, then a network TCP booster is impossible. Mobility and multi-path routing also may make boosters less effective, because messages may not always pass through the same pairs of boosters. Some boosters, such as an FZC booster, will work well in this environment, because the boosters do not share any state. For booster with state, such as the ARQ booster, there would be performance degradation.

5 Implementation and Experimentation

We added protocol boosters support to FreeBSD and Linux operating systems for the i386 (Intel) architecture. The main difference between the two is that the Linux version supports booster insertion/deletion from the run-time environment which allows dynamic loading/ removal of booster modules into network elements. Booster support resides in the IP portion of the TCP/IP networking stack. Every IP packet processed is examined to detect if it is a member of a boosted communication channel. If boosting is specified, the packet is handed to the appropriate booster/debooster. Our implementations currently identify boosted channels by source and destination IP addresses; however, we can add stronger filtering based on other information, such as port number.

An FZC booster has been designed and inserted into the FreeBSD infrastructure [2]. For each boosted channel, the FZC booster caches, then immediately forwards, each data packet it receives: whether the packet is from an upper layer protocol or the IP Forwarder. The only modification to each data packet is that the FZC booster overwrites the IP packet's 16-bit identification field with a sequence number, allowing the decoder to know the packet's position information. This does not change the end-to-end UDP datagrams or TCP segments.

After receiving k packets (k is defined per channel) on a given channel, the cached packets are zero-padded to the size of the largest packet in the cache. Also each packet's size and protocol type are appended to the packet's tail. The transmitter performs a FZC matrix multiplication over the payload, padding, and appended tail of the k packets. The h overcode packet payloads produced by the FZC encoder are then prepended with an IP header and a booster header. This IP header contains a prototype field identifying it as a protocol booster packet and a sequence number in the 16-bit identification field. The booster header contains the type of booster (FEC Booster), the value k, and the sequence number of the first of the k packets. The h packets are then transmitted toward the same destination as the data packets.

The FZC debooster at the receive side caches incoming data packets and immediately forwards them to either an upper layer protocol or towards their eventual destination. Overcode packets are also cached, but are not forwarded. Packets are released from the cache when: (1) k data packets are present, (2) the received data packets plus parity packets equals k, or (3) when the cache occupancy dictates cache content replacement. Only in situation (2) are the matrix computations performed to generate the missing data packets.

To assess the effectiveness of this booster arrangement we deployed it in a simulated wireless environment running UDP. The amount and types of errors on wireless networks depend upon link conditions and link error control. In general, however, errors are not random but come in a burst of consecutive bits. Whether the packet errors are also bursty depends on the size of packets and average burst error lengths. Based on the actual packet loss results obtained in satellite experiments [11] we constructed two basic packet error models: random and bursty. These error models were used in a loss module, placed in the IP receive path as depicted below, that can delete packets based on the chosen error model.

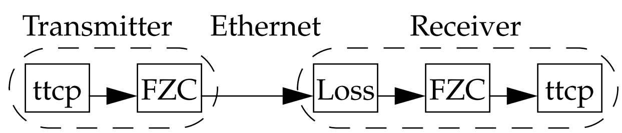

Figure 3 Experiment Setup

As our application we used the public domain Test TCP program (**ttcp**), with the UDP option. We ran ttcp between two machines on a dedicated ethernet. As shown in Figure 3, both ends ran the FZC booster and the receiver also ran the loss module. The experiments used a packet size of 512 bytes, with 1000 packets per trial, and a block size of 20 packets. As expected, the results [2] show that with no parity packets the effective packet loss is roughly the same as the loss rate defined in the loss module, and that increasing redundancy decreases the effective packet loss. Although burstiness further increases the network packet loss, the FZC booster is still effective. In fact, larger block sizes make the difference between bursty loss and random loss negligible.

6 Summary

We propose a new method for protocol design based on incremental, dynamic construction of protocols on an as-needed basis. The elements of these protocols, protocol boosters, can be composed in a Tinkertoy-like manner, to form a family of protocols. The methodology motivates optimistic protocol design, where protocols are designed assuming the best case, and add additional functionality on an as-needed basis. This is in direct contrast with building for the worst-case and optimizing by finding common fast-paths through the protocol implementation. Prototypes of the booster design methodology have been implemented [2]. The performance was sufficiently encouraging that a larger-scale design is being investigated.

Protocol boosters can be viewed as a step towards the fully programmable infrastructure proposed by a number of researchers under the rubric of "Active Networks." [15]. While many of the problems are the same (e.g., robust end-to-end behavior, security for systems into which boosters are loaded, etc.), a key advantage of boosters is that they can easily be injected into today's systems without a wholesale change in the network infrastructure. In that sense, they offer an early test of the promise of active networks.

The protocol booster methodology offers some exciting possibilities for accelerating the evolution of protocols, for changing the economics of protocol development, and for creating useful "hybrid" protocols which have "just enough" support for the heterogeneity actually encountered. We view the use of protocol boosters to build protocols as an optimistic approach to protocol design. Applications implement application-to-application protocols assuming an ideal world. Protocol boosters add functions on an as-needed basis to provide this ideal world; when the world really is ideal (e.g., homogeneous workstations on a LAN) no overhead is incurred.

REFERENCES

[1] T. Alanko, M. Kojo, L. Laamanen, K. Raatikainen, and M. Tienari, "Mobile Computing Based on GSM: The Mowgli Approach,"IFIP'96: Mobile Communications, Canberra, Australia, September 1996.

[2] D. Bakin, W. Marcus, A. McAuley, T. Raleigh, "An FEC Booster for UDP Application over Terrestrial and Satellite Wireless Networks," International Mobile Satellite Conference (IMSC 97), Pasadena, CA, June 1997.

[3] A. Bakre and B. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," Proceedings IEEE 15'th Annual Conference of Distributed Computer Systems, Vancover, Canada, May 1995.

[4] H. Balakrishnan, S. Seshan, R. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," ACM Mobile Computing and Networking Conference (Mobicom '95), Berkeley, CA, November 1995.

[5] D. Clark, V, Jacobson, J. Romkey, H. Salwen, "An Analysis of TCP Processing Overhead," IEEE Communications Magazine, pp. 23-29, June 1989.

[6] D. Clark and D. Tennenhouse, "Architectural Considerations For A New Generation Of Protocols," Proceedings of ACM SIGCOMM, pp. 200-208, September 1990.

[7] D. C. Feldmeier, "A Data Labelling Technique for High-Performance Protocol Processing and Its Consequences," Computer Communications Review (SIGCOMM '93), Vol. 23, No. 4, pp. 170-181, October 1993.

[8] P. E. Green, "Computer Network Architectures and Protocols," Plenum, New York, 1982.

[9] N. C. Hutchinson and L. L. Peterson, "The x-Kernel: An architecture for implementing network protocols," IEEE Transactions on Software Engineering, 17(1), pp. 64-76, Jan. 1991.

[10] V. Jacobson, "TCP/IP Compression for Low-Speed Serial Links." RFC 1144, February 1990.

[11] A. J. McAuley, D. S. Pinck, T. Kanai, M. Kramer, G. Ramirez, H. Tohme, and L. Tong, "Experimental Results From Internetworking Data Applications Over Various Wireless Networks Using a Single Flexible Error Control Protocol," Fifth WINLAB Workshop: Third Generation Wireless Information Networks, East Brunswick, New Jersey, April 26-27, 1995.

[12] A. J. McAuley, "Reliable Broadband Communication Using a Burst Erasure Correcting Code," Proceedings of ACM SIGCOMM, pp. 287-306, September 1990.

[13] J. C. Mogul, "The case for Persistent-Connection HTTP," in Proceedings, 1995 SIGCOMM Conference, pp. 299-313, 1995.

[14] J. H. Saltzer, D. P. Reed, & D. D. Clark, "End-to-end Arguments in System Design," Proceedings of the 2'nd IEEE International Conference on Distributed Computing Systems, pp. 509-512, April 1981.

[15] J. Smith, et al., "SwitchWare: Accelerating Network Evolution (White Paper)," {http://www.cis.upenn.edu/~ jms/white-paper.ps}, 1996.