The CHART System: A High-Performance, Fair Transport Architecture Based on Explicit-Rate Signaling*

Jack Brassil
HP Labs

Rick McGeer
HP Labs

Raj Rajagopalan
HP Labs

Puneet Sharma
HP Labs

Praveen Yalagandula
HP Labs

Sujata Banerjee
HP Labs

David P. Reed
HP Labs

Sung-Ju Lee
HP Labs

ABSTRACT

TCP/IP is known to have poor performance under conditions of moderate to high packet loss (5%-20%) and end-to-end latency (20-200 ms). The CHART system, under development by HP and its partners under contract to the US Defense Advanced Research Projects Agency, is a careful re-engineering of Internet Layer 3 and Layer 4 protocols to improve TCP/IP performance in these cases. The CHART system has just completed the second phase of a three-phase, 42-month development cycle. The goal for the 42-month program was a 10x improvement in the performance of TCP/IP under conditions of loss and delay. In independent tests for DARPA at Science Applications International Corporation, the CHART System demonstrated a 20x performance improvement over TCP/IP, exceeding the goals for the program by a factor of two. Fairness to legacy TCP and UDP flows was further demonstrated in DARPA testing. We describe the CHART System as a set of five interacting services and protocol improvements which act together to make TCP/IP robust under conditions of loss and latency, and we describe and detail the test regime and performance results.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Packet-switching networks*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol architecture (OSI model)*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network Monitoring*; C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks—*Internet (e.g., TCP/IP)*

General Terms

Algorithms, Design, Experimentation, Measurement, Performance, Security, Standardization

Keywords

Quality of Service, Flow-based Routing, Explicit-Rate, Virtualization, Overlay

*This work was supported in part by DARPA Contract N66001-05-9-8904 (Internet Control Plane). This work Approved for Public Release, Distribution Unlimited

1. INTRODUCTION

The Control for High-Throughput Adaptive Resilient Transport (CHART) project is a 42-month effort between HP Labs and its partners to add an intelligent control plane to the Internet. In particular, the CHART project introduces a suite of new signaling protocols between network systems and end hosts, and among network systems, to provide enhanced TCP performance, quality-of-service guarantees to applications such as IP television and voice over IP, and a high degree of network information between network elements and end hosts and among network elements. The need for an intelligent control plane arises from the reality that the Internet Protocol stack arose in an era of limited applications (bulk data transfer), low capacity, and expensive memory and processing. The resulting design deliberately sacrificed Quality-of-Service(QoS) to provide best-effort service that makes no guarantee on the quality of routes. Further, Internet performance depends on the transmission rate of the end host. Since today's endpoint protocols, particularly TCP, originated in an era of expensive processing and memory, they were designed to infer network state from measurements that could be made at the endpoints. In particular, packet loss was taken as an indicator of network congestion, and as a result TCP implementations reduce transmission speed in the presence of packet loss. Thus, end-to-end performance degrades rapidly in response to relatively minor loss of link quality.

The CHART program is designed to address this problem through two principal architectural innovations to the Internet Layer 3 and Layer 4 protocols:

1. Fine-grained signaling and sensing within the network infrastructure to detect subtle and transient link failures and route around them; and
2. Explicit agreement between end hosts and the routing infrastructure on transmission rate, which will permit the end hosts to transmit at the agreed rate independent of loss and delay.

CHART achieves these innovations through the following set of loosely-coupled services:

1. A network-wide real-time network monitoring service

(the "Chart Sensing Infrastructure" or CSI) that is reliable, efficient, scalable, and secure.

2. A new generation of network elements which monitor and control *flows*, and which explicitly communicates available bandwidth to new and existing flows.

3. A new TCP driver, TCP-Trinity, which implements the explicit-rate protocol on the end hosts.

All of the infrastructure software overlay elements (the network sensing overlay and the software routing overlay) run on a dispersed collection of commodity IA-32 computers running a containerized version of the Linux Fedora Core operating system. These computing nodes may be dedicated to supporting CHART, or may be a shared decentralized computing overlay such as *PlanetLab* [13].

It is important to note that explicit signaling encompasses, in many ways, a fundamental redesign of the basic premises of the network, offering a number of new opportunities and requiring new devices and capabilities. The fundamental new opportunity is the ability to overlay *dynamic virtual circuits* onto packet-switched networks: channels with guaranteed properties and isolation. In this sense, it is a marriage of virtualization technology with the network: this offers a degree of isolation difficult to achieve in packet-switched networks; and since it is used only for applications which demand high quality-of-service, it still permits high bulk throughput on the common channel. The fundamental new services and capabilities required are the abilities familiar in the rest of information technology; particularly, the ability to explicitly allocate and manage network resources much as we today manage computation, memory, and storage. Various attempts have been made before to introduce this capability into the network, from ATM to the Resource ReSerVation Protocol (RSVP)[3], to MultiProtocol Label Switching (MPLS)[17] to Differentiated Services (DiffServ)[2]. CHART is a deep exploration of the concept of explicit signaling, its effects on network performance, requirements on the endpoints and the in-network elements, and new security requirements.

Explicit signaling means explicit resource allocation, and a contract between the network and an endpoint and application. Implicit in this is the notion of *priority* and *preemption*. High-priority requests must be able to pre-empt resources allocated to other users. Moreover, the network must be able to offer certificates that requested resources were actually delivered to the requester.

The need for pre-emption and allocation immediately implies the need for *authentication* and *authorization*.

The remainder of this document describes the CHART system and results and is organized as follows. Section 2 gives an overview of our solution, and the explicit bandwidth-signaling protocol that is a core element of the solution. The signaling protocol imposes significant security requirements. In particular, the signaling protocol implements pre-emption and prioritization, critical for high quality-of-service applications; as a result, signaling requests must be *authenticated*. Section 3 gives an overview of the security architecture which permits safe, undeniable authentication and ensures that access is restricted to the requester. Section 4 details the hardware and software network elements that monitor existing bandwidth and grant or deny the request.

Once this solution is in place, a number of desired capabilities become possible. In particular, high-performance explicit-rate transport, fair to existing traffic, becomes possible. In section 5 we detail the design and implementation of a high-performance explicit-rate TCP and UDP driver and TCP proxy. Section 6 describes our global sensing architecture. We conclude with a description of the results of the program and a short summary.

2. DESCRIPTION OF THE SOLUTION

TCP/IP was designed in an era of low-loss, low-latency connections, and where the capacity of network elements to do computation was very limited. The resulting protocol suite used in-band sensing of latency and packet loss as its primary tool to determine transmission rate. Further, there was no mechanism to allocate and regulate bandwidth on a flow-by-flow basis. This resulted in a protocol suite which was quite effective, efficient, and fair in an environment of reliable links and in an application space dominated by bulk data transfer, which characterized the Internet for the first generation of its existence.

This use case – bulk data transfer over reliable, low-latency lines – was the sole use case motivating the Internet's original design. However, today's Internet is characterized by a wide variety of use cases, including QoS-intensive applications such as voice, video, and gaming, and high-latency, lossy links. The original design assumptions of the Internet are therefore no longer valid in an increasing variety of situations. In particular, the Internet's early choice to sacrifice guaranteed Quality of Service for individual flows in order to ensure high bulk throughput and link utilization now forces over-provisioning to deliver services with high QoS demands [5] and loss and latency are no longer reliable indicators of network congestion.

The CHART program is an exploration of the use of two fundamental tools to approach this problem: direct network sensing through a pervasive sensing plane and explicit in-band signaling between network elements and end hosts to assign transmission rates for individual flows.

Explicit signaling for transmission control has been proposed before. An out-of-band signaling approach is at the heart of the Resource ReSerVation Protocol (RSVP) [4] and in-band signaling was proposed by Dukkipati et. al. in the Rate Control Protocol [7]. The CHART program extends these ideas in three significant ways:

- The ability of end-host applications to *request* specific rates from the network and for the network to grant these requests, critical for high-QoS applications, as well as the ability for the network to simply grant the best available rate based on current traffic conditions.
- The ability of a sensing overlay to perform the role of the network in granting flow requests, through a system of *overlay flow management*.

- Design of a new generation of network flow elements, commercially available from Anagran Networks, to perform the network tasks of allocating bandwidth to and managing individual flows.

At the heart of the innovation is an Explicit-Rate option to the Internet Protocol (IP), implemented as an IPv6 option header and as a prefix to the TCP data field in IPv4. The ER option permits an application and the network to agree on a specific transmission rate, periodically renewed. This permits (i) the network to allocate bandwidth to specific flows, ensuring quality of service to specific applications without over-provisioning and (ii) the endpoints to simply transmit at the agreed rate, without resorting to the complex control-loop used in standard TCP implementations. The result is the removal of dependence on latency and loss in determining network performance. We have implemented a Linux-based explicit rate-aware protocol stack, "TCP-Trinity," which enables end hosts to accelerate data transmission by bypassing the slow-start and congestion avoidance phases of data transfer.

2.1 Signaling Objectives

The objective of the QoS mechanism is to permit the necessary resources to be allocated to a flow as it traverses the network. To permit real-time allocation and rapid renegotiation in the face of changing network conditions, the signaling scheme is in-band, and requires support in the participating network elements. This will require hardware or microcode support, a burden when compared to out-of-band software-based schemes such as RSVP. The resource request and response messages are incorporated into application data packets, allowing the QoS requirements to be established during the flow initiation from sender to receiver and back. This signaling scheme can be used to set the rate, burst tolerance, pre-emption priority, delay priority and charging direction for a flow.

Signaling strategies for IPv4 and IPv6 use the same QoS structure. For IPv4, the document specifies a QoS Structure to be added to the first data packet of a flow, requiring a unique DiffServ code-point. The QoS structure immediately follows the transport header. For IPv6, the same QoS information is added as a hop-by-hop option.

For both IPv4 and IPv6, each participating network element in the path examines the QoS Structure and agrees to or adjusts the rates requested to the rates it can support. If any of the rate parameters have been changed by the network elements in the flow path, this is communicated back to the sender by the receiver.

The QoS defined within this signaling structure can support four general types of service. The first is a fully guaranteed rate flow, which implies no oversubscription of network resources. The second is a maximum rate flow, which allows some oversubscription but virtually no packet loss. The third is a variable rate flow, where available rate is combined with a minimum rate guarantee. The fourth is an available rate flow, one that can jump start the Transmission Control Protocol (TCP) to the highest rate the network can support, eliminating slow-start problems. In the available rate case the capacity available based on network congestion is fed back to the sender very rapidly at all times. This will help to differentiate congestion problems from channel errors (measured in bit error rates), permitting the sender to then optimize packet error control without confusing it with congestion.

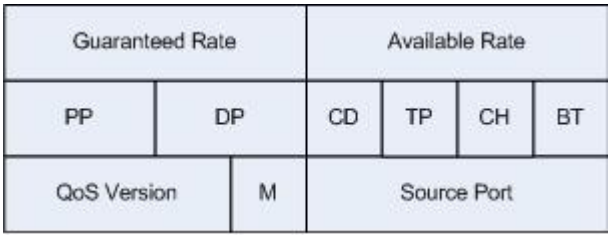

Figure 1: QoS Header Structure

2.2 The Explicit-Rate Protocol

The TIA-1039 [15] Explicit-Rate protocol is best understood as a transport-neutral mechanism by which the end hosts and network elements can negotiate a renewable explicit rate. In the ideal case, any transport protocol should be able to use this facility. Moreover, since available bandwidth along a path is the minimum of the available bandwidth along each link, rate negotiation must take place between the end host and each network element along the path. The net result is an IP-layer design: the rate request and response ought to be in a header explicitly accessed by routing elements (and thus should not be higher than Layer 3). Further, since Layer 2 headers can be entirely replaced and rewritten by routing elements, the explicit rate option should not be in the Layer 2 header: if it is, each routing element requires that its neighbor preserve a specific piece of Layer 2 information.

The result is an IP option: option fields must be supported by all IPv6 routing elements, and thus in a v6 network one can rely on the preservation of option fields throughout the network. The option fields are shown in Figure 1.

The fields depicted in Figure 1 give a total of 12 bytes for the option header. Available Rate(AR) and Guaranteed Rate(GR) are 2-byte fields, encoded as 14-bit floating point numbers with a zero bit and a reserved bit. The lower 5 bits of the floating point number are the exponent, and the nine high-order bits are the base, giving a range of values from $(1, 2^{41})$. They are in units of kilobits per second, permitting a rate of approximately 4 Terabits per second, or about 40 lambda in today's highest-end technology.

The *PP* and *DP* fields are used to indicate to the network element the *pre-emption priority* (lower-priority flows are to be pre-empted to accommodate the rate request of this flow) and *delay priority* (this flow is to be accelerated past lower priority flows) of this flow. The *CD* field is an internal field which indicates the direction and status of the option field, so that elements can correctly interpret this signal as a request or response. Status of the request/response is also indicated. The *TP* field indicates type of flow (TCP, UDP, etc) and the *BT* field indicates the ability of the flow to tolerate deviation from the agreed rate. The *CH* field supports charging information and field *QoS Version* specifies the specific version of the protocol in use. The *M*, or *modi-*

fied field, is set to 0 by a requester and set to 1 if any field was modified during a request or renegotiate phase, and set to 0 and not changed on a response.

Since network conditions change, rate requests are renegotiated every 128 packets or one second, whichever is more frequent. This procedure and rate are similar to those used in Available Bit Rate (ABR) signaling in ATM [1]. If we consider a Voice-over-IP flow sending a 3 kilobit packet every 50 ms, we see that rate renegotiation occurs at the maximum interval of every second, or after about sixty kilobits have been transmitted. Even if we consider the case of maximum packet size of 12 kilobits (assuming an MTU of 1500 bytes), rate negotiation occurs no less frequently than the transmission of every megabit.

3. SECURITY

Security enhancements in a commercial product are, by definition, for protecting value, either in the context of a business model or of information assets in field deployment. The common commercial business model for QoS services (such that those supported by the TIA-1039 protocol) is predicated upon the service provider providing superior connectivity (higher bandwidth, lower latency, less jitter, etc.) to some target consumer. Threats to this business model include commonplace threats such as theft of paid service as more exotic threats such as service disruption by prejudicial agents. The addition of security mechanisms to any system will incur additional cost, thus the goal of security is to enhance the design so that the appropriate security can be achieved at a reasonable cost. We note that the scope of security here is limited to the QoS signaling aspect and does not cover securing the legacy aspects of the service. For example, the security of the data flow itself is outside the scope of this project. Accordingly, we have attempted to provide a reasonable level of control and resistance against theft-of-service attacks on the provision and maintenance of QoS service with appropriate protocols and cryptography-based security mechanisms.

Mindful of the fragile nature of security solutions, we impose some meta-requirements on our design: (i) the security design must adhere to and reuse standards as much as possible, (ii) security must be minimal and the design must allow for future extensions and additions, and (iii) security functionality must be, to the extent possible, offloaded from functional devices such as proxies and routers onto separate security-specific entities. In the following, we describe the specific security requirements as well as how we address these requirements in CHART. The security requirements for TIA-1039-based service can be divided into three categories: user authentication/authorization, control signal protection, and key management.

User authentication/authorization (AA): The primary defense against the threat of theft of service in most systems is user AA. We note that this requirement holds only when users need to be identified or differentiated for the purpose of restricting access to service or different levels of service. Even though the baseline use case for CHART does not include differentiated services, specific target uses such as the NCS GETS use case require that specific users (first responders) need to be able to access specific services preferentially. Furthermore, note that because the TIA 1039 design works wholly within TCP and UDP flows, the concept of a user is outside the protocol and consequently user AA is required solely for manageability of the protocol. In the CHART project, the decision point for user AA is the AA server, the enforcement point is the entry point of the flow (which may be the ingress flow router or the client proxy), and user credentials are X.509 certificates issued and signed by a universal Certification Authority. The user on the client presents (over https) his X.509 certificate to the AA server of the administrative domain of the current network. The AA server validates the credentials and creates a time-bound authorization for that client machine's IP address based on administrative policy stored in the AA server. The enforcement points periodically download current authorizations from the AA server. When a new flow request is received by an enforcement point, it automatically accepts or rejects the request based on the authorizations downloaded into that enforcement point from the AA server. If the client is authorized, standard TIA 1039 processing proceeds, otherwise the request is either dropped silently or accorded a "free" service level that is set by policy. A similar (mirrored) image of this protocol occurs at the server end with the egress router and server proxy.

Control Signal Integrity Protection: Flow control integrity is necessary to protect against the threats of service disruption and theft. Flow signals that are sent as part of the TIA 1039 headers may be modified by malicious agents that have access to the packets in transit such as on legacy IP routers. Detection of such unauthorized modification is achieved using hash-based message authentication codes (HMAC) from the IPSEC standard, which currently specifies SHA-256 as the hash algorithm. Since the signal can be modified legitimately by 1039-capable routers on the path, this HMAC is created on the entire 1039 header plus some additional fields (source and destination IP Addresses) and validated between two successive 1039-capable agents. HMACs require a secret key be established between the sender and receiver. This secret key is generated on demand by the AA server and downloaded into each of the pair of communicants. HMAC creation and validation is performed on every 1039 hop, between client proxy and ingress router, between successive 1039-capable routers, and between the egress router and server proxy. Note that confidentiality of the control signal is not a requirement and thus we do not perform any encryption of the header. Similarly, HMACs can only detect modification but cannot recover the original message. Furthermore, our security mechanisms cannot prevent an adversary from modifying behavior by selectively dropping packets. All entities have to be configured to ignore or drop all 1039 signals that do not have valid HMAC entries. In the prototype, the HMAC is implemented using the Open SSL library.

Key Management: HMACs require a shared secret (symmetric) key between sender and receiver. When a sender S needs to send a HMAC'ed packet, it looks up its "next-hop" table to determine the receiver R for that destination. If the R cannot be found, it queries the AA server for R. To create the HMAC, S can use a valid cached secret key for the pair (S,R) or contact the AA server with a request for a HMAC key for the pair. The AA server generates and serves

the required key using a strong pseudorandom generator if a valid cached key is not available. Analogously, when a packet is received by R, it can validate the HMAC using a cached key or request the AA server for the key. The AA server will reply with the key previously generated for the pair. If the intended receiver in the signal was not R, then the HMAC will fail as well. R then notifies the AA server using another query that includes S, R, and the destination address. In response the AA server will update the next-hop table so that S will be notified in the next update. All keys are refreshed by the AA server according to administrative policy. The AA server also validates user credentials, which are signed with a universal certificate authority whose public key is known to the AA server. Functionally, there will be one AA server per administrative domain though there may be multiple copies for redundancy and to minimize latency. Co-ordination between AA servers of two different domains is done by hand on a bilateral basis. All communications with the AA server are via https with mutual authentication. In the prototype, the AA server is implemented using Apache.

4. FLOW MANAGEMENT ELEMENTS

The new mechanism required by the CHART program is the ability of the network to route and manage flows, not individual packets. This requirement implies the need for a new generation of network appliance which can manage flows.

Informally, a flow is the stream of packets from one user to another that forms a specific file transfer or conversation. Formally, a flow is uniquely identified in IPv4 by the five-tuple: source address, destination address, source port, destination port, and protocol. In IPv6 a three-tuple (flow label, source address, destination address) is used to identify the flow. Experimentally, the average flow is short-lived, containing only about 14 packets.

Anagran networks has developed a hardware element, the FR-1000 flow manager, which can support an aggregate bandwidth of 48 Gbps through four line cards, each of which can support 12 1Gbps or one 10Gbps port. The Anagran flow managers maintain performance statistics on each outgoing link. Flow statistics are exported via the industry standard *NetFlow* protocol to a computer running network management software (e.g., cflowd, flow-tools) to facilitate network-wide traffic engineering.

Flow managers support network operation at high utilization. Since the rate of each flow is controlled with a flow manager, the total rate being fed to a trunk is also controlled. By measuring the load on an output trunk and feeding this information back to all the input ports, the total utilization of an output port can be controlled to within 5%. We anticipate that under a normal TCP traffic mix the average utilization can be maintained above 80%, which compares favorably to the relatively lower average utilization of trunks in US carrier networks.

Flow managers achieve QoS guarantees by allowing equalized load balancing among users, rapid TCP rate feedback, and guaranteed rate, loss, and delay for voice and video flows. This type of flow management is necessary in broadband networks to achieve high quality voice and video traffic. For our purposes the QoS of a flow is described by the parameters Guaranteed Rate (GR), Available Rate (AR), Burst Tolerance (BT), Delay Variance (DV), and Precedence or Pre-emption Priority (PP).

The flow manager controls the QoS of all active flows by establishing QoS parameters for each flow upon setup, and adjusting these parameters over the life of a flow.

The initial QoS parameters for a flow are based on the requested QoS for the flow and the QoS of the available paths. The requested QoS is determined either by an explicit request or by a rule based on a combination of fields in the first packet of a flow (i.e., a given DiffServ code may indicate Voice-over-IP (VoIP), which corresponds to QoS parameters of GR equaling 82 kbps and DV of less than 10 milliseconds). The available QoS of the paths that the flow could be routed over is determined by measuring the QoS parameters of all egress ports for a variety of flow classes (i.e., VoIP, video, and data).

A sender requests a flow with available rate, guaranteed rate, delay, and precedence parameters specified. The first flow manager reduces the available rate to what it can support. Each subsequent flow manager in the path does the same until the message reaches the receiver. The receiver then reflects the agreed path rates to the sender, and the sender confirms them across the network to release over-commitments. A sender equipped with an Explicit-Rate aware TCP stack (see Section 5) can immediately increase its sending rate to the agreed rate, in this example 32 Mbps. This rate can be maintained until the network needs to adjust the rate (up or down) due to cross-traffic.

4.1 An Explicit-Rate-Aware Software Overlay

The Explicit-Rate mechanism can extend to paths that do not consist exclusively of flow managers in two scenarios:

1. The estimated *available bandwidth* along a sub-path of legacy routers between two flow managers on a given path would be used by the upstream flow manager to compute the AR value corresponding to the "virtual" link between the two flow managers.
2. A software overlay flow manager along a given path could compute the AR value for an outgoing overlay link similarly to a flow manager and write the AR information onto QoS packets traversing the link.

In both cases, the available bandwidth information for a given Internet path is provided by the CHART sensing system.

The second case corresponds to what we call an *Explicit-Rate (ER) overlay*. Figure 2 depicts how the ER overlay allows the software overlay routing system to emulate the available rate feature provided by the flow managers. IP QoS flows with GR = 0 can be supported by an ER overlay even in the absence of the hardware-based flow managers. Within the ER overlay, a given overlay node computes an AR value corresponding to each outgoing overlay link using available

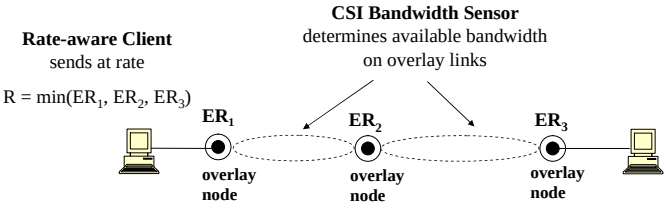

Figure 2: An Explicit-Rate overlay.

bandwidth estimates obtained from the sensing infrastructure and internal packet queue lengths. The AR value for an overlay link is reported to the flows traversing the link in accordance with the TIA QoS signaling protocol [16].

The algorithm employed by the overlay node to compute the AR value need not be the same as the algorithm used in the hardware-based flow managers. We are currently experimenting with an ER control algorithm based on the NEC rate control algorithm for Available Bit Rate (ABR) services in ATM networks [9]. This algorithm does not require maintenance of per-flow state, has good performance characteristics, and is stable. Besides supporting the IP QoS flows (with zero GR value), the ER overlay can also support Quick-Start TCP, an enhancement to TCP proposed within the IETF [8]. Quick-Start TCP allows a flow to jump to a large initial congestion window size via a signaling mechanism similar to the TIA IP QoS protocol, but does not provide a full-fledged congestion control mechanism.

5. ENDPOINT ELEMENTS

The explicit-rate protocol described above was designed both to permit flows which require a guaranteed rate (e.g., voice and video flows) to reserve it, and also to significantly improve the performance of best-effort TCP flows via the available rate (AR) mechanism. We have developed a new TCP stack, *TCP-Trinity* to exploit this feature.

Under the available-rate mechanism, a TCP-Trinity sender requests the available rate and receives the response in a single round trip time. The sender then immediately jumps to the explicit rate, and continues to transmit at that rate even in the presence of packet loss and latency variation. The available rate request is renewed every 128 packets, permitting the network to signal changes in available bandwidth to senders rapidly, giving an explicit mechanism to avoid congestion.

We have developed a client system capable of exploiting the availability of an end-to-end rate reservation through a network of flow routers. An initial ER-aware X86-based PC running the FC 4 operating system is used for the host. The end system uses the proposed TIA QoS protocol [16]. The TCP/IP stack has been modified to support a "fast-start" option invoked by a Linux ioctl() system call. When "fast-start" is invoked for a connection the modified TCP/IP stack will disable the TCP slow start algorithm and send at the rate specified by the QoS parameters specified in the fast-start ioctl call. The TCP-Trinity stack uses selective acknowledgment and retransmission (SACK) for error recovery on QoS-enabled connections. In the absence of ER information from an ingress router, the client TCP-Trinity stack operates in a conventional fashion.

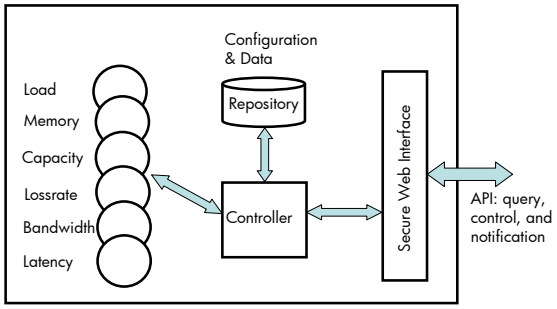

Figure 3: S^3 sensor pod.

6. S^3

The primary goal of the Scalable Sensing Service (S^3) is to provide an easily accessible, accurate, and scalable network monitoring service to monitor end-to-end network metrics between participating nodes. Another important goal is to provide flexible interfaces to applications that enable sharing of measurement data across multiple applications and allow monitoring at finer time scales as required for specific applications. Such a service eases network management tasks such as root-cause analysis, better VoIP routing, and the placement of computing, network, and storage resources. Also, it allows end applications to make better decisions based on the network status. For example, a client in a Content Distribution Network can choose the best server for which the path between the client and the server has the highest available bandwidth compared to other servers.

In [23], we describe the S^3 architecture with three components:

(i) Sensor pods: Run at end nodes, these are collections of sensors that expose interfaces for various types of sensor invocations,

(ii) Sensing information backplane: Provides a programmable middleware for aggregating and disseminating the sensor data, and

(iii) Inference engines: Infer end-to-end network metrics without performing all-pair measurements but through performing only a few measurements and inferring the properties of all other end-to-end paths [18, 21].

As described in the introduction, the CHART system depends on a sensing infrastructure to monitor the network state and feed that information to other components of the system. The monitoring and control component of CHART uses sensor pods to perform periodic measurement of available bandwidth and latency metrics from each participating node to the neighboring nodes in the CHART routing substrate. Other components of the CHART system, such as the TCP-Trinity driver, that uses the available bandwidth measurements to decide the congestion window size instead of observed packet losses, subscribe to these measurement streams.

Sensors	Purpose
PING	Measures latency to a specified destination
TRACEROUTE	Collects number of hops and latency on the network path to a destination
PATHRATE [6]	Measure capacity of the network path to a destination
TULIP [12]	Measures error rate of the network path to a destination
SPRUCE [19], PATHCHIRP [14]	Measure available bandwidth on the network path to a destination

Table 1: A subset of network sensors in S^3 deployment on PlanetLab.

A sensor pod is a web service-enabled collection of lightweight measurement and monitoring sensors that collect information at a machine. This information spans both network properties such as connectivity to the Internet, latency to some other machine in the system, bandwidth to another machine, and machine attributes such as machine's current CPU load, free memory, and number of processes. These sensors gather information actively (e.g., send some packets on a network link to detect available bandwidth) or passively (e.g., infer the current RTTs of a link from communication pattern of a TCP connection on the same link). Simple sensors can also be created to extract existing SNMP MIB data from different network elements.

As shown in Figure 3, along with sensors, each sensor pod has a Data Repository (DR) to store information measured by different sensors, a Configuration Repository (CR) to store sensing configurations that are used to determine which sensor to invoke, how frequently to invoke, how long to invoke, what parameters to supply on sensor invocation, and how to store the information returned by the sensor. Also each sensor pod has a controller that coordinates the invocation of sensors based on the configuration information provided to the sensor pod. Sensor pods expose interfaces for both querying the data in DR and for setting new configurations in CR through web service API. Exposing sensor pod interfaces as a web service in our architecture enables sensor composition — new sensors can be easily built by composing information generated by some existing sensors.

A sensor pod optimizes the number of sensor invocations (to minimize network communication and computation costs) by analyzing sensing configurations supplied by different applications for a unique sensor. For example, if two applications need to measure the latency to the same destination machine but at different periodic rates, say once in 6 seconds and once in 10 seconds, then our subsystem invokes the sensor at the higher rate (once in 6 seconds) and stores that as an answer for both sensing configurations. Also if multiple applications invoke the same sensor for one-shot measurement and if those invocations overlap, then the sensor pod performs only one measurement and returns the same value to all requesting applications. Thus, our sensor pod design effectively eliminates the redundant monitoring traffic incurred by applications if each conducts its own measurements. We are developing a measurement scheduler to limit the measurement overhead under a specific fraction of the network capacity (e.g., 1% of the link capacity).

We have built a prototype of the S^3 modules and deployed them on on the PlanetLab testbed. This deployment has been running since January 2006. We spent significant engineering effort on multiple fronts to get to the current stage where the entire system is automated and requires almost zero maintenance and is easy to extend. Our current implementation has a wide variety of sensors, some of which are listed in Table 6, that leverage several open source network monitoring tools for measuring various network path metrics (latency, number of hops, available bandwidth, bottleneck capacity, and loss rate). We are currently measuring all-pair network metrics periodically.

We pull the measurement data from the sensor pods on all nodes to a central node to provide the global views to other researchers by making this data available online, and also to archive the data for post-analysis of the Internet behavior. A snapshot of the all-pair capacity and available bandwidth metrics updated about every 4 hours is available at the following website: `http://networking.hpl.hp.com/s-cube/PL`.

The data from the S^3 deployment on PlanetLab was useful for other research projects internal and external to HP Labs. This data was used in substantiating the benefits of bandwidth-aware routing in the overlay networks [10]. We were able to quantify the relationships between different network metrics [22] to explore if a low-cost metric can be monitored to deduce changes in a high-cost metric. We also provided the data to researchers from Purdue, Microsoft Research, Cornell, UIUC, UArizona, NTT Japan, and UCSD.

7. EXPERIMENTAL RESULTS

The CHART system has been designed to increase end-to-end bulk file transfer throughput performance by a factor of 10 in a challenging operational WAN environment. All elements of the CHART system have been implemented and undergone extensive testing on the PlanetLab, the Emulab network emulation system at the University of Utah, and in a DARPA testbed at Science Applications International on a set of test networks developed by a testing team at SAIC.

To verify the performance of control plane enhancements, DARPA has created a set of three network test scenarios. Each scenario introduced up to 10 separate network impairments or defects, such as an excessively high packet loss on a link, or temporary link failure. Bulk data transfer tests were performed using ftp.

The CHART solution was tested by SAIC on their testbed in two four-week sequences of tests in the summer of 2006 and 2007. The mean result showed a TCP performance improvement over legacy Linux 2.6 by a factor of 20, exceeding the goals for the entire program by a factor of 2.

Here, we demonstrate results obtained in Emulab[11] testing. Unless otherwise noted, all results were obtained over a nominal 100 Mb/sec capacity link. Times were measured using iperf[20], using 60-second timed transmissions. Bandwidth was reported by iperf and measured at the receiver. All transmissions used 1-MByte initial windows at both sender and receiver, set by `sysctl`.

The problem of standard TCP is illustrated in the chart in Figure 4. Here, the decline in throughput of standard TCP

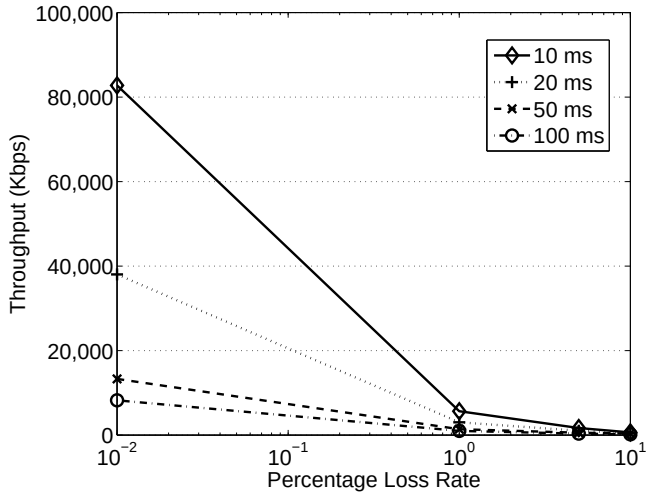

Figure 4: Performance of Legacy TCP under Loss and Latency.

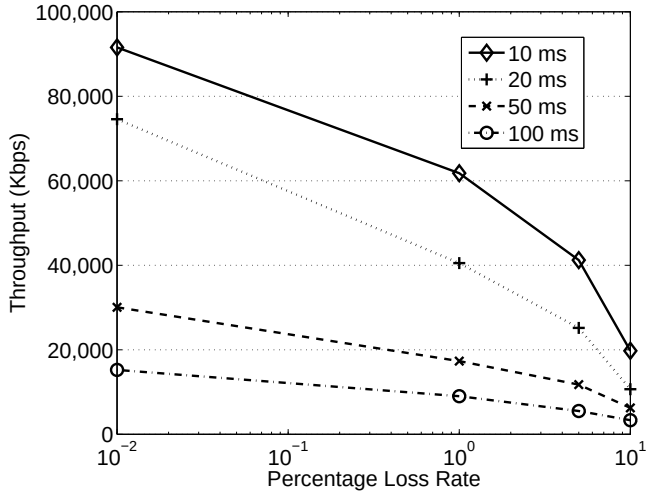

Figure 5: Performance of CHART TCP under Loss and Latency.

with increasing latency is shown dramatically, at every loss level. Note particularly the effect of loss, even at very small latencies. At low latency, 10% loss, throughput is only about 600 Kb/sec – about equivalent to a reasonable cable modem. At 100 ms latency, this degrades to about 180 Kb/s.

The contrast to the CHART system is shown in figure 5. While some decline in performance due to increasing latency is apparent, as is some decline due to loss, the degradation of performance in response to increasing latency and loss is far more graceful. In particular, 10% loss at no delay still admits a throughput of over 65 Mb/s, and even at 100 milliseconds latency throughput is still a respectable 2 Mb/s.

These relative performance figures translate into a significant performance increase for CHART TCP, particularly at high latency and loss. The relative performance is shown in figure 6. As can be seen from the figure, the CHART TCP =-Trinity driver demonstrates a significant performance increase over legacy TCP at all legacy/delay points, with dra-

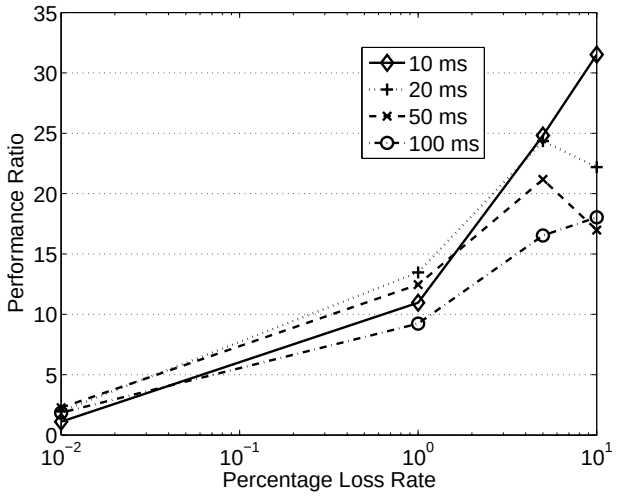

Figure 6: CHART TCP/Performance Ratio under Loss and Latency

matic increases on the 5% and 10% loss curves.

Throughput is of course important, but it begs an even more important question: fairness. TCP degrades quickly with increases in loss and latency because loss is regarded by TCP as a proxy for congestion: TCP controls congestion in the network by reducing transmission speed. It is therefore easy to make TCP robust in the face of loss: simply ignore loss. Of course, there may be consequences for congestion control. A skeptic might argue that all we have demonstrated is that it is easy to drive from the Lincoln Memorial to the Capitol in two minutes at 9 o'clock AM on a Monday morning, as long as one is relatively unconcerned about collateral damage.

We therefore have a greater burden. Not only must we show that the CHART stack improves performance, we must prove that it abides by the first law of the Internet: first, cause no congestion.

In order to investigate the fairness properties of CHART TCP, we first sought to determine whether CHART TCP would reduce its channel consumption to accommodate other flows. In order to test this, first we ran two legacy flows against each other, at sufficiently low loss and delay that each could individually fill the pipe. We then ran a single CHART flow against a legacy flow under the same conditions, and finally ran two CHART flows against each other. In all cases, we calculated the fairness of one flow with respect to the other as the complement of the total throughput taken by the flow. We also measured the total throughput of all flows.

The results are shown in figure 7. Results were compared to a single legacy flow, labeled here as baseline. Fairness is shown here as the percentage of total throughput taken by the *other flow*. A score of 1.0 indicates that the flows were perfectly balanced, whereas a score of under 1.0 indicates that this flow took more than its share. We also measured total throughput expressed as a percentage of the theoretical

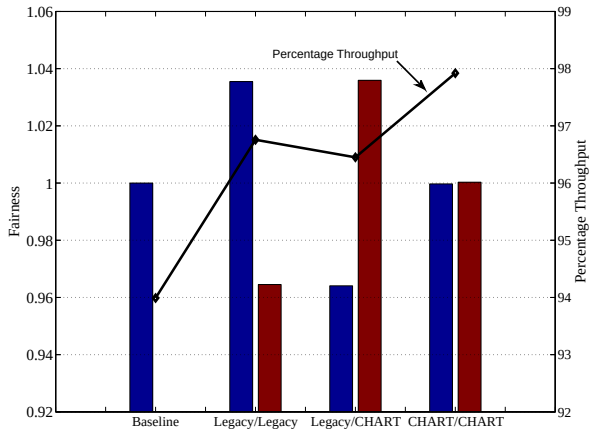

Figure 7: Fairness and performance of Flows in Fairness Experiment

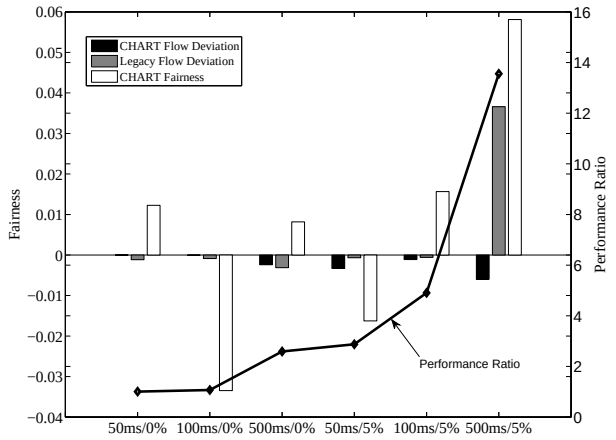

Figure 8: Fairness and performance ratio of Legacy and CHART flows in different Latency/Loss cases.

throughput of the line, with a theoretical maximum of 100%.

The single Baseline flow achieved 94% of theoretical maximum, and perfect fairness by definition (1.0). Conversely, when CHART (QoS) was run against legacy TCP. it achieved a fairness score of 1.04 – it gave the legacy flow about 8% more bandwidth than it took – and the two flows together achieved 97% of theoretical throughput.

As can be seen from the graph, the CHART flow was the fairest flow of all, yielding a surplus of 8% of bandwidth to a legacy flow. In the CHART/CHART case, bandwidth was perfectly balanced between the flows. Further, this combination had the highest throughput, at 98% of theoretical maximum.

This experiment demonstrates that the CHART solution is fair between CHART flows, and is fair to legacy flows: indeed, fairer than legacy flows are to each other. However, this case investigated the area where the latency and delay were sufficiently small that legacy flows could fill the pipe. This still leaves the case where the CHART flow can easily fill the pipe, but the legacy flow cannot: will the CHART flow reduce its throughput to make room for legacy?

In order to test this, we ran two sets of legacy flows in parallel, and computed the aggregate bandwidth. We then ran a set of CHART flows against a set of legacy flows, and determined the ratio of bandwidth consumed by the legacy flow against the CHART flow against the mean of the two legacy flows taken together. This ratio was taken as the fairness of the CHART flow set to legacy flows. We also measured the deviation of each flow set about its mean, and used this as a measure of the internal fairness among a flow set.

The results are shown in in Figure 8. Note that all but one case the deviation from the mean of the CHART flows was smaller than the deviation of the legacy flows, indicating greater internal fairness in the CHART flowset in each case (perfect fairness here would be a deviation of 0.0).

Similarly, in four of the six cases legacy throughput actually increased when competing against the CHART flows, shown by the "CHART fairness" series. This series measures the ratio of a legacy flowset competing against a CHART flowset. A positive score indicates an increase in bandwidth when competing against CHART; a negative score, a decrease in bandwidth. Further, CHART did not lose its performance advantage when competing with legacy flows, even though it remained fair. The performance ratio of the CHART flowset to the legacy flowset is also shown in Figure 8.

8. CONCLUSION

The CHART system offers a new approach to the problem of improving end-to-end throughput performance, combining in-band signaling, decentralized performance measurement and monitoring, and the strategic deployment of a new generation of advanced flow managers. The combination of network sensing and explicit signaling has been shown in preliminary tests to increase throughput in the presence of bad links by an order of magnitude or more for both pure hardware and pure software implementations. A unique and compelling aspect of CHART is that it permits the gradual introduction of new flow management technology on the network, where overlay nodes on the existing infrastructure provide the immediate benefits of explicit signaling. Hardware flow managers may be judiciously added to a network to accommodate transmission rates up to 10 Gbps. Overlay nodes also facilitate the introduction of new network applications.

CHART's approach not only solves the end-to-end performance problem sought by the DARPA Internet Control Plane program, it also lays a foundation for future network-wide applications. The reason for this is that once a computational overlay is deployed – in this case to support software flow management – the overlay can be exploited for a variety of other novel, decentralized applications.

9. ACKNOWLEDGEMENTS

The authors thank a large number of people who have contributed over the years to the success of the CHART program. We particularly acknowledge the unflagging support

of the US Defense Advanced Research Projects Agency, and thank the program manager who conceived the Control Plane program, Dr. Timothy Gibson (COL, US Army, Ret.), and his able SETAs. We thank our supportive management over the years: Frank Pietryka, Patrick Scaglia, Norm Jouppi, Susie Wee, John Apostolopoulos, and Mike Freeman. The CHART program owes an especially deep debt to the Emulab testbed. We wish to take this opportunity to mourn the tragic and premature loss of one of the genuinely great men of systems computer science, and our close friend, Jay Lepreau. We encourage the systems community to continue to support Jay's beloved and enduring legacy, Emulab, and his students and successors: Mike Hibler, Eric Eide, and Rob Ricci. We thank Craig Soules for careful reading of this paper and insightful comments.

10. ADDITIONAL AUTHORS

Andy Bavier, Princeton University,
Larry Peterson, Princeton University,
Stephen Schwab, Sparta Inc.,
Larry Roberts, Anagran Inc.,
Alex Henderson, Anagran Inc.,
Bob Khorram, Anagran Inc.,
Shidong Zhang, George Mason University,
Soonyong Sohn, George Mason University,
Brian Mark, George Mason University,
Christian Heiter, HP,
John Spies, HP, and
Nicki Watts, HP.

11. REFERENCES

[1] ATM Forum. ATM Traffic Management Specification Version 4.0 <af-tm-0056.000>, Apr. 1996.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. IETF RFC 2475, Dec. 1998.

[3] B. Braden, L. Zhang, B. S., S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. IETF RFC 2205, Sep. 1997.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard), Sept. 1997. Updated by RFCs 2750, 3936, 4495.

[5] J. Brassil, B. Mark, R. McGeer, S. Schwab, P. Sharma, and P. Yalagandula. The case for service overlays. In *Preparation*, 2008.

[6] C. Dovrolis, P. Ramanathan, and D. Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Transactions on Networking*, 12(6):963–977, December 2004.

[7] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown. Processor sharing flows in the internet. In *Thirteenth International Workshop on Quality of Service (IWQoS)*, june 2005.

[8] A. Jain, S. Floyd, M. Allman, and P. Sarolahti. Quick-start for tcp and ip. In *Internet-draft draft-ietf-tsvwg-quickstart-00.txt*, May 2005.

[9] A. Kolarov and G. Ramamurthy. A control-theoretic approach to the design of an explicit rate controller for abr service. *IEEE/ACM Trans. on Networking*, 7(5), October 1999.

[10] S.-J. Lee, S. Banerjee, P. Sharma, and P. Yalagandula. Bandwidth-Aware Routing in Overlay Networks. In *Proc. IEEE INFOCOM*, 2008.

[11] J. Lepreau. Emulab network emulation testbed. `http://www.emulab.net`.

[12] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet Path Diagnosis. In *Proceedings of the SOSP*, Oct 2003.

[13] PlanetLab Consortium. Planetlab. `http://www.planet-lab.org`.

[14] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficient available bandwidth estimation for network paths. In *Proceedings of the PAM 2003*, La Jolla, CA, April 2003.

[15] L. Roberts. QoS Signaling for IP QoS Support. TIA 1039, July 2005.

[16] L. Roberts. *TIA TR-34.1.7 Working Group (IP on Satellite)*, chapter TIA TR-34.1.02/12.04.05: QoS Signalling for IPv6 QoS Support. Telecommunications Industry Association, 2005.

[17] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. IETF RFC 3031, Jan. 2001.

[18] P. Sharma, Z. Xu, S. Banerjee, and S.-J. Lee. Estimating network proximity and latency. *ACM Computer Communications Review*, 36(3):41–50, July 2006.

[19] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of the ACM IMC 2003*, Miami, FL, October 2003.

[20] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. Iperf version 2.02.

[21] R. Wouhaybi, P. Sharma, S. Banerjee, and A. Campbell. Minerva: Learning to Infer Network Path Properties. In *Proc. IEEE INFOCOM*, 2008.

[22] P. Yalagandula, S.-J. Lee, P. Sharma, and S. Banerjee. Correlations in End-to-End Network Metrics: Impact on Large Scale Network Monitoring. In *Proc. IEEE Global Internet Symposium*, 2008.

[23] P. Yalagandula, P. Sharma, S. Banerjee, S.-J. Lee, and S. Basu. S^3: A Scalable Sensing Service for Monitoring Large Networked Systems. In *Proc. SIGCOMM Workshop on Internet Network Management*, 2006.