Towards an Active IP Accounting Infrastructure

Franco Travostino
Nortel Networks Technology Center
600 Technology Park
Billerica, MA 02174
travos@nortelnetworks.com

***Abstract*- Traditional IP accounting infrastructures cannot withstand the shock waves produced by voice over IP integration, increasingly large accounting data volumes, adaptive pricing schema that reflect resource utilization, and real-time requirements on accounting applications such as billing and traffic monitoring.**

To suit tomorrow's networks, the Active IP Accounting Co-processor Environment (AIACE) argues that the number of accounting tasks performed at the network nodes and at the accounting servers be fluid and not necessarily known a priori. The AIACE infrastructure introduces the notion of an active accounting co-processor coupled with the network node. This co-processor performs custom operations on accounting data exactly where the accounting data is being generated. For instance, the co-processor can efficiently manage pre-paid debit card semantics, reflect settlements among providers, apply fraud detectors, or enable real-time accounting data mining.

The AIACE infrastructure dynamically pushes intelligence into network nodes and applies Active Networks themes to network management. Isolated from any problematic interference with end-to-end data paths, AIACE can set IP accounting free to exploit the power of open and active network nodes.

I. INTRODUCTION

In traditional IP accounting, network nodes (e.g., routers, switches, firewalls) collect data regarding the network traffic that flows through them. From accounting-illiterate network nodes, the data get periodically uploaded into centralized accounting servers. The latter synthesize the unwashed accounting data off-line, and make the outcome available to accounting applications such as billing, traffic monitoring, capacity planning, and auditing. Increasingly, the effectiveness and efficiency of traditional IP accounting are being challenged by the following trends:

- Accounting data volumes growing linearly with respect to the bandwidth available to end users' traffic;
- High-value, high pay-off traffic coexisting with low-cost, low-value traffic within the same commodity infrastructure;
- Ubiquitous presence of IP (e.g., for both data and voice);
- Quality of Service classes for tariffs, billing, monitoring procedures, which will include, for instance, real-time billing to selected customers;
- Adaptive pricing schema designed to reflect and promote resource utilization (as opposed to flat administrative agreements among providers);
- Pay-for-what-you-use models wherein service providers advocate to themselves the freedom to customize the set of communication resources and computation resources that are being monitored or billed for (e.g., bandwidth as well as, say, the cycles of a crypto-accellerator);
- Interoperability challenges among vertically integrated vendor-specific solutions for network management.

The Active IP Accounting Co-processor Environment (AIACE[1]) questions traditional IP accounting at the very foundation. The new AIACE infrastructure argues that the number of accounting tasks performed at the network nodes and at the accounting servers must be fluid and not necessarily known a priori. That is, network nodes cease to be accounting-illiterate and, to the contrary, effectively pre-process accounting data to an extent that the recipient accounting servers can control. In this model, accounting plug-ins are the elemental units of accounting processing; they can nest into the AIACE co-processor and perform specific accounting tasks at the network node on behalf of an accounting server.

Among things, the new accounting-savvy network nodes can eagerly aggregate the accounting data of flows meeting pre-set affinity criteria, manage pre-paid debit card semantics, reflect settlements among providers, apply fraud detectors, enable real-time accounting data mining, and custom signal accounting servers to meet the deadlines of accounting applications.

With accounting-savvy network nodes, AIACE's goal is to greatly increase the "goodput" between network nodes and accounting servers. In turn, this can improve the responsiveness, scalability, and dependability of accounting applications (e.g., real-time billing, traffic monitoring, fraud detection) to the benefit of service providers or network operators.

[1] AIACE is the Latin transcription for Ajax, the ancient Greek hero who distinguished himself during the Trojan War and earned the prize of Achilles' mighty armory.

AIACE poses new requirements on how network nodes are engineered. These requirements adhere to the vision that in the next decade network nodes will no longer be vertically integrated systems. That is, manufacturers will pass on to service providers and entrepreneurs the freedom to formulate the high-value services for which they have identified a market. The customizable AIACE accounting infrastructure wants to take advantage of these new opportunities.

The paper proceeds with a brief description of current IP accounting technologies, followed by an overview section that puts AIACE in focus. In this overview, a selection of the opportunities created by the node-resident accounting co-processor is presented. The paper continues with the definition of the AIACE architecture, in terms of building-blocks for both the co-processor and the co-processor aware accounting server. The concepts presented in these sections come together in two sample accounting applications—custom billing for application flows and high-confidence traffic monitoring— which capitalize on the AIACE co-processor opportunities.

II. Traditional IP Accounting

IP accounting concerns the collection of usage metrics relating to IP networks.

In traditional IP accounting (Fig. 1), hardware or software meters are placed into or near by network nodes (e.g., routers, switches, firewalls). RMON/RMON2 [15], OC3MON [2], SMON [16] are well known types of meters. Meters can capture events (e.g., a PDU in or out) with various levels of granularity and intrusiveness. With or without aggregation, the meters' output makes it into traffic reports that are transferred from network nodes to accounting servers. The protocol for this transfer is referred to as the accounting protocol. SNMP [19], RADIUS [14], NetFlow [11], OSA [12] are examples of protocols used as accounting protocols. The node-resident software that makes a network node interact with an accounting server typically does not qualify as a Turing machine.

Accounting servers [3] collect, aggregate, correlate, remove duplicates, and store accounting data. This off-line elaboration of accounting data is known to be a lengthy process, whose duration is often quantified in terms of hours or days rather than seconds or minutes. The outcome of the elaboration is eventually made available to accounting applications ("applications" throughout the paper) such as billing services, capacity-planning tools, traffic monitoring, or auditing services.

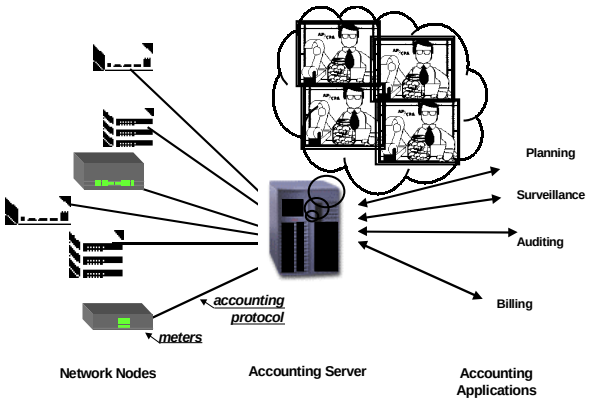


Fig. 1. Traditional IP Accounting

Much as individuals and corporations use different styles to maintain their books, IP accounting can hardly be framed into a "one-size-fits-all" model. IP accounting requirements may differ based on:

- Which data is being captured by meters, and which layout is most appropriate for accounting data records; whether Quality-of-Service (QoS) metrics such as delay or jitter matter. Minimally, an accounting record can be thought of as a 5-tuple[2] record augmented with number of packets and bytes forwarded in a unit of time. Records may optionally be structured into unidirectional or bi-directional flows;
- Where accounting data is being sampled—e.g., at ingress or egress interfaces, at edge network nodes only (carrier) or at specific internal hubs to account for internally generated traffic (intranets);
- Why accounting data is being sampled—e.g., for billing, intrusion detection, or traffic monitoring;
- When and how often accounting data is being transferred from network nodes to the accounting server(s);
- How the accounting data is being transferred—e.g., with SNMP, RADIUS, securely or in the clear—or stored—e.g., to volatile or non-volatile storage. The accounting protocol may be synchronous or asynchronous, standard or proprietary.

The industry trends enumerated in the introduction will continue to deliver many solutions along these five dimensions. The time lag between new features in network nodes (e.g., VLAN), the resolution by standard bodies (e.g., SMON), and actual market penetration by standards is unacceptably large. It encourages non-interoperable solutions and proprietary drifts.

[2] A 5 tuple consists of: protocol number, source and destination IP address, source and destination port number.

Furthermore, there is a fundamental contrast between meters generating increasingly large data volumes at the network nodes (left side of Fig. 1), and applications using increasingly sophisticated contexts for customer care (right side of Fig. 1). If applications continue to have exclusive access to their context, they will not be able to be effectively supported by network nodes or accounting servers in accomplishing their mission. This becomes particularly evident when real-time applications are being considered—i.e., applications whose output must be dependably delivered according to published timings and accuracy. For applications to behave in real-time, and with sub-minute latencies, the mining of relevant accounting data must also happen in real-time. This implies that real-time properties must be provisioned throughout the whole assembly chain, from the meter to the final application output.

For instance, a user may want to subscribe to a new usage-based billing service that shows how costs are building up in real-time, with a latency of ten seconds. This service requires the service provider to aggressively capture and process the accounting data concerning this type of user. In turn, the service provider may need to renegotiate the contracts with its providers and possibly subscribe to new accounting/billing services of theirs.

III. AIACE Overview

The AIACE programmable accounting co-processor is a Turing machine operating directly at the network node. With its plug-and-play software modules (i.e., the accounting plug-ins), the co-processor has the potential to overcome the limitations observed in traditional IP accounting. The anticipated AIACE contributions fall into the overlapping categories of:

- Effectiveness improvements (e.g., real-time accounting data mining for custom billing or high confidence traffic monitoring);
- Multiple efficiency scenarios (e.g., to achieve a data reduction on accounting's own wire traffic at the expenses of richer hardware or more CPU cycles being invested in the accounting co-processor);
- Accuracy tradeoffs (e.g., to manage accounting data overloads in real-time by considering financial implications of data drops);
- Increased interoperability through just-in-time binding between meter formats and accounting protocols.

A. Separating Accounting Signal from Noise

Given a set of application requirements, it must be possible to classify the unwashed accounting data into a useful portion—the signal—and an accessory portion—the noise. The signal represents the amount of accounting data that is necessary and sufficient to meet an application's goal. This process is referred to as accounting data mining.

In the aforementioned real-time billing example, the accounting data pertaining to the set of real-time customers only qualifies as signal. The remaining accounting data that pertains to unqualified customers is noise. Let us further assume that the real-time IP billing service comes with a published accuracy of one USD (which would typically be compensated by a monthly flat rate to limit the provider's exposure). Should accounting and/or billing incur into overload within a given real-time sampling period, the definition of signal can be restricted further to those real-time-enabled accounting records with traffic figures greater than a given traffic threshold. These records will also pass the billing threshold of one USD.

The foremost AIACE objective is to amplify and expedite accounting's signal over noise, for definitions of signal and accuracy that are not necessarily known a priori, and can be traced back to an application's own context. These definitions are dynamically supplied by the application or service provider who exercise the AIACE infrastructure. By specifying the nature of the signal and its accuracy, an application can actually pass some of its context on to network nodes, in exchange for better support by network nodes.

As for the accounting noise, the closure of the various application-specific requirements determines whether to batch it for later processing, or to suppress it all together at the network node.

Electrical engineers are well familiar with the unmatched signal-to-noise gains that come from positioning an amplifier as close as possible to a RF antenna. These gains can be formally verified. With accounting plug-ins, AIACE wants to prove a similar theorem: An accounting plug-in (the amplifier) collocated with the network node (the antenna) can greatly boost the IP accounting's own signal-to-noise ratio by operating exactly where the accounting data is being captured (Fig. 2).

B. Co-processor Opportunities

The accounting plug-ins that load into and execute within the AIACE co-processor may perform the following activities (a non-exhaustive list):

- Structure PDU traces into IP flows. Aggregate IP flows belonging to lower-than-IP logical partitions (e.g., VLANs) or higher-than-IP sessions (e.g., H.323 [7], SIP [18] sessions) using node-local knowledge (e.g., IEEE 802.1Q tags, a H.323 proxy in a firewall);

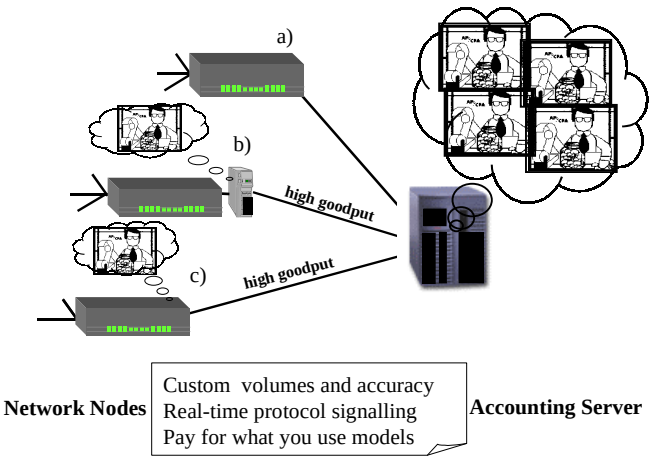


Fig. 2. The AIACE architecture uses an active IP accounting co-processor operating at network nodes b) and c). In b) the co-processor is loosely coupled with the network node, whereas in c) the co-processor is fully integrated with the network node; a) is a traditional accounting-illiterate node.

- Eagerly mine the accounting data that appeal to applications with real-time requirements. This may entail custom extensions to the accounting protocols between co-processor and accounting server;
- Realize accounting semantics required by pre-paid debit card and toll-free semantics. In the former, a counter is set to a pre-defined value and decreased whenever qualifying traffic passes by, until there are no credits. In the latter, the accounting record is no-op'd because no cost will be incurred;
- Eagerly recognize settlements among accounting domains, and dispatch accounting data accordingly. Likewise, prepare for departmental chargebacks. These activities may entail an interaction between the co-processor and the node-local routing table database;
- Actively manage the accounting accuracy. In the case of billing applications, this amounts to transposing the one USD accounting rule into IP accounting[3]. That is, flows that do not account for some minimal entry figure of traffic are assimilated into noise;
- Tag PDUs that have already been accounted for and thus should not be counted two or more times by other network nodes within an intranet (Within intranets, the risk of double counting PDUs arises from the fact that traffic may be generated internally to the intranet; thus, the sampling of accounting data at the edge nodes only is necessary but not sufficient);
- Dynamically adapt the X layout for meters to the PDU layout mandated by the Y accounting protocol;
- Distinguish accounting records that belong to different virtual private networks (VPNs); apply accounting policies that are VPN specific;
- Custom extend the set of monitored or billable resources (e.g., to include the cycles of a hardware crypto-accellerator, or the number of wired routing table entries into the forwarding database);
- Minimize fraud exposure with fraud detectors being remotely driven and dynamically adjusted by the accounting server;
- Segregate and document accounting data pertaining to UDP flows in order to apply punitive measures (because they do not back off in case of congestion);
- Enable real-time trend analysis by establishing and monitoring statistically-relevant meters. Tune up meters' sampling according to feedback metrics supplied by applications and its algorithms (e.g., genetic algorithms);
- Advertise temporarily unused resources back to the accounting server to enable last-minute discount pricing schema;
- Selectively encrypt accounting data to guarantee confidentiality.

C. Active Networks and the End-to-End Argument

While Active Networks [21] concepts appeal to a large audience, the application of Active Networks themes to network management and to IP accounting in particular has thus far enjoyed little attention.

The AIACE architecture qualifies as a discrete application of Active Networks to network management. Accounting plug-ins are explicitly dispatched off-line, and outside of the PDU's payload bay. Some literature would refer to accounting plug-ins as Active Services. The points of convergence between the AIACE infrastructure and the Active Networks approach can be summarized as follows:

- Accounting plug-ins are actual programs (i.e., data as well as code) that nest into the node-resident accounting co-processor;
- Accounting plug-ins take advantage of information that exclusively lives in the network (this concept of affinity reminds of other real-life activities that make particularly good sense when performed in place … e.g., ABS car brakes work effectively because they pump brakes "in place");
- The space of possible accounting plug-ins may be vast, sparse, and evolving, so as to make operator-assisted loading/unloading of software impractical and error prone;

[3] One USD is an acceptable degree of accuracy in medium-to-large accounting practices in the United States.

- Plug-ins' loading and activation may be triggered from either side—i.e., from the network node side or from the accounting server side.

The AIACE architecture is not affected by the often-heard criticism of Active Networks violating the end-to-end argument. In AIACE's case, the two ends are the co-processor and the accounting server. Whereas traditional IP accounting (Fig. 3) exhibits asymmetric stacks at each end, and in particular uses a minimal stack at the network node end, AIACE makes the two ends interact effectively through more balanced, extensible protocol stacks. Plug-ins can be reciprocated in the stacks at each end.

D. Implementation Strategies

Virtual machines marry the accounting server's need to dispatch accounting plug-ins into the network nodes, with the heterogeneity of the recipient network nodes.

It is well known that real systems result from striking a balancing act between contrasting forces. The state of the art for network nodes has had virtual machines succumbing to efficiency forces. Increasingly, however, hardware resources are enabling a different balancing act—an act that allows a targeted, introductory use of virtual machines. These virtual machines are being engineered as hardware or software co-processors (as in [9]). The actual packaging of the co-processor—be it in hardware or software—is outside of the scope of this paper.

Starting from these hardware-rich configurations, there are great opportunities to align the software evolution curve of network nodes with the evolution of accounting server systems and end-systems in general.

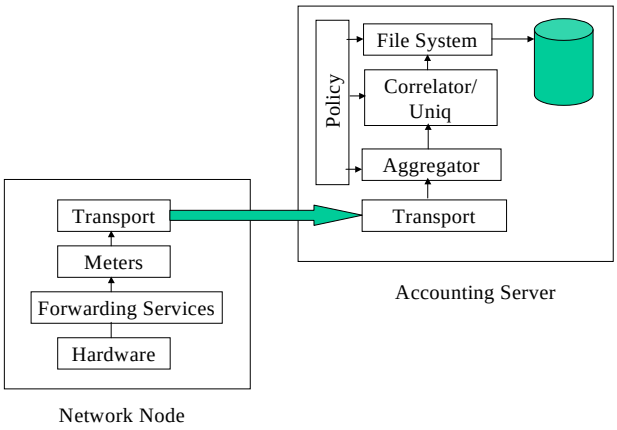


Fig. 3. Building blocks in traditional IP accounting.

In this context, virtual machine technologies that are already widely popular within end-systems are preferred to proprietary virtual machines for the co-processor. For the nomadic and plug-and-play nature of accounting plug-ins, the virtual machine must be extensible with security devices, platform neutral code representation, feature discovery, and remoting architecture facilities.

Today, the Java technology best matches this description of virtual machine. Java is not an exact match, though. Java's current shortcomings include poor performances and lack of service guarantees. Just-in-time compilation techniques mitigate the former. The workload patterns observed by accounting plug-ins—few instructions repetitively applied to large date sets—are a reasonably good ground for just-in-time compilation techniques and their peephole performance optimizations. As for service guarantees, Java Virtual Machine real-time extensions and resource-safety enhancements [5] deliver a Java platform that is predictable and robust to theft of resources (including the case of resource theft by buggy or misconfigured code). While the Java technology is the chosen mean to AIACE, other mechanisms—including the ones based on native code—can also be used to realize the AIACE infrastructure.

In AIACE implementations based on multiple languages and bindings, the alignment with the Active Networks Encapsulation Protocol (ANEP) will guarantee multi-language support. As for compatibility, accounting server implementations can choose a dual-stack approach to communicate with network nodes powered by the AIACE accounting co-processor as well as with legacy network nodes. To dual-stack servers, accounting plug-ins look like optional boosters of accounting data processing. At the server, the plug-ins' output rejoins mainstream processing after discounting the processing steps already carried out by the plug-in.

III. The AIACE Architecture

A. Co-processor Organization

The fundamental building blocks of the AIACE accounting co-processor are (Fig. 4):

- The *virtual accounting device*. This layer provides a platform-independent representation of accounting resources—e.g. accounting data records, timing services, unique identifiers, hash functions, lookup algorithms, etc. When the co-processor is based on Java, the virtual accounting device layer represents an extension to the Java runtime;
- The accounting data *classifier*. The classifier applies rules and actions to accounting data records. A rule is best thought of as the conditional clause of an "if" statement; an action corresponds to the code in the

"then" or "else" blocks. This classifier borrows terminology and techniques from PDU classification [4].

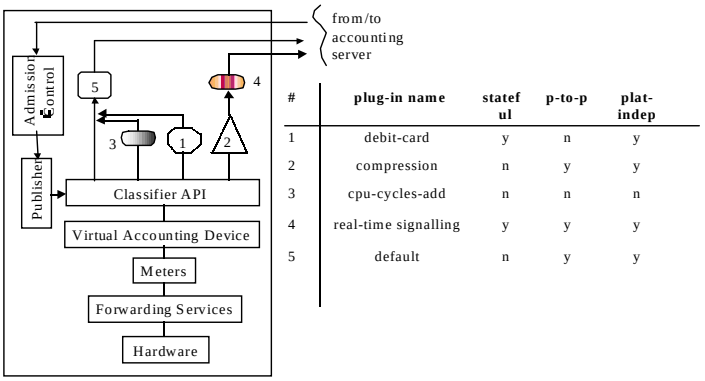
- The *accounting plug-in*. The plug-in represents a primitive action upon an accounting record or a set of accounting records. In other component-based technologies, this plug-in would be referred to as a micro-protocol [10] or a Java Bean;
- The plug-in *framework*. With this framework, multiple accounting plug-ins "stack up" and incrementally refine a given accounting record or a set of accounting records;
- The *demand loader*. The loader is responsible for loading accounting plug-ins on demand from the accounting server, or another neighboring trusted accounting co-processors;
- The *admission controller*. The controller performs capacity-based and credentials-based checks before a new rule or plug-in can be instantiated;
- The *publisher*. The publisher is a portal through which other entities external to the co-processor (e.g., the accounting server as well as local application proxies, intrusion detectors) can register new rules and/or actions.

Accounting plug-ins lend themselves to several classifications.

First, accounting plug-ins can be labeled as hard-state or soft-state within the co-processor. Unlike hard-state, soft-state is recoverable and can be unilaterally collected by the co-processor upon timeout expirations (e.g., like in a LRU cache). Consider an accounting plug-in that realizes pre-paid debit card semantics. The plug-in is loaded into the co-processor with an initial credit; this credit is decreased by the co-processor when qualifying traffic passes by. This plug-in represents hard-state, in that the state it carries (the actual credit) is a precious resource (i.e., it may be recovered only through batch analysis of the unwashed traffic by the accounting server, and not in real-time).

Likewise, the privacy plug-in is a hard-state plug-in, in that it contains encryption keys used to provide privacy to the transfer of records to the accounting server.

The splitter plug-in is an example of soft-state plug-in. It aggressively applies administrative settlements by dispatching accounting records to providers in different administrative domains (and thus under the jurisdiction of other accounting servers). Should the splitter plug-in be collected away, the accounting server will incur more processing to apply settlements. Soft-state plug-ins are best thought of as boosters, in that they optionally boost IP accounting processing. The accounting server must be prepared to act in their behalf in the case that these plug-ins were collected away.



#	plug-in name	stateful	p-to-p	plat-indep
1	debit-card	y	n	y
2	compression	n	y	y
3	cpu-cycles-add	n	n	n
4	real-time signalling	y	y	y
5	default	n	y	y

Fig. 4. The architecture of the node-resident co-processor, with 5 sample accounting plug-ins (contrast with left side of Figure 3).

To the contrary, hard-state plug-ins are a precious resource and, together with selected accounting records, must be check-pointed to stable storage (e.g., onto the node's NVRAM).

Secondly, accounting plug-ins can be classified based on their local or peer-to-peer significance. The aforementioned pre-paid debit card plug-in has local significance, in that its action is entirely consumed within the accounting co-processor. To the contrary, a compression plug-in has peer-to-peer significance, because it expects a matching de-compression plug-in to be applied at the accounting server side. In general, all the plug-ins that affect the transfer of data from the co-processor to the accounting server need to be matched peer-wise.

Thirdly, accounting plug-ins may be platform-neutral or platform-dependent. While most plug-ins are expected to be platform-neutral, the AIACE infrastructure admits platform-dependent plug-ins. These plug-ins express a deep interaction with the system—e.g., to track CPU cycles—that would otherwise be hard to define through extensions to the virtual accounting device.

B. Server Organization

The AIACE IP accounting server includes logic to support co-processor operations, and to sustain a peer-to-peer conversation with the co-processor. While the co-processor operates within a resource-constrained environment, the accounting server is typically blessed with a large amount of computational resources and storage.

The fundamental building blocks of the AIACE architecture residing at the accounting server side are (Fig. 5):

- The *plug-in store*. This unit hands off plug-ins to the co-processor upon request from the demand loader. Additionally, it maintains the database and name space of the known plug-ins;
- The *policy writer*. With the writer, the accounting server talks to the co-processor's publisher and expresses new policies in terms of rules, actions, and plug-in names;
- The accounting *plug-ins* that realize a server side matching functionality. Those plug-ins with peer-to-peer significance need in fact to be instantiated both at the co-processor and at the server;
- The plug-in *framework*. As described for the co-processor, this framework provides the scaffolding and mechanisms for accounting plug-ins to compose on the server side;
- *APIs* to the accounting server's core functions, such as aggregators, correlators, and ultimately to the billing services (should billing be the ultimate goal of IP accounting). With these APIs, it is possible to fast-track accounting data records and thus capitalize on the work previously done by the accounting co-processor at the network node.

C. Dynamics

The accounting co-processor pulls accounting plug-ins from the accounting server on demand (much as virtual memory pages are loaded on demand from a file system server). The co-processor's demand for a plug-in can be explicit or implicit.

An explicit plug-in demand-loading cycle occurs when a new action corresponding to a verified rule is being triggered. An explicit loading cycle also occurs when an accounting plug-in was formerly instantiated and later was collected due to extended inactivity. In this case, the co-processor asks the accounting server for the plug-in to be re-instantiated (or simply recovered, depending on the stateful versus stateless nature of the plug-in).

An implicit demand loading cycle is started when new accounting records are created at the co-processor. Periodically, the co-processor publishes a directory of newly created records to the accounting server. The server reacts to this report by providing rules, actions, and plug-ins to be applied to those accounting records that are eligible for node-local treatment. In some circumstances, the accounting server may delegate the authority to spell out rules, actions, and plug-ins to other entities, such as application proxies local to the network node (e.g., a H.323 proxy) and formerly loaded with users' profiles.

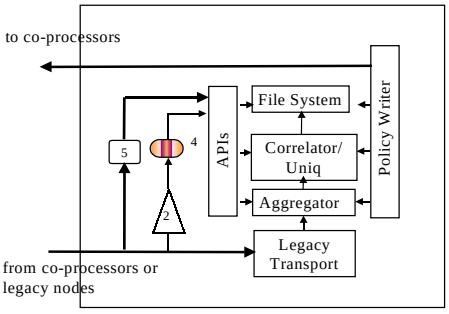


Fig. 5. The architecture of an accounting server that communicates with co-processors as well as legacy nodes (contrast with right side of Figure 3). Plug-ins' legend is in Fig. 4.

The AIACE architecture does not currently mandate that the creation of new accounting records and the instantiation of new accounting plug-ins be synchronized.

To expedite both explicit and implicit loading of plug-ins, the co-processor may elect to locally manage a LRU cache of plug-ins.

V. Sample Applications

A. Custom Billing for Application Flows

A H.323-based [7] multi-media conference is taking place among users in different administrative domains and security enclaves; the participants select the premium Quality of Service (QoS) class. The domain where User A lives is protected by a firewall under control of User A's Internet Service Provider (ISP); this firewall is also accounting-enabled to support the ISP's business model. From the ISP, User A wants to have a real-time account of the expenses incurred while conferencing. Depending upon the actual content, quality, and cost of the conference, User A may elect to switch to other QoS classes or ISPs.

The conference participants interoperate through the H.323 protocol. This protocol is known to defeat the traditional 5-tuple filtering by the firewall. The root cause for this is that a unique H.323 session comprises several UDP/TCP sub-flows using dynamic UDP/TCP ports. The firewall in User A's domain obviously cannot open up all ports to all traffic entering into the domain.

The firewall counters the elusive H.323 session by establishing a H.323 proxy. A H.323 proxy is an application

proxy (L7[4] proxy) that runs inside the firewall and mediates H.323 traffic. Inbound H.323 traffic is explicitly addressed to the proxy, and from the proxy is forwarded to User A; outbound H.323 traffic is transferred from User A onto the proxy, and from the proxy is sent to the outside destinations.

From an IP accounting standpoint, the ISP's accounting system must:

1) Pick the accounting data records relating to the H.323 conference out of the whole set of accounting data. Meters only document L3 and L4 information (i.e., IP addresses and TCP/UDP port numbers). The correlation of L3/L4-based accounting data into a unique accounting record for the whole H.323 session is non trivial (as observed in the case of mere firewall filtering);

2) Verify that the aggregate accounting data records from step 1 report values of jitter and latency that are consistent with User A's service contract;

3) To fully characterize the costs inherent in the H.323 conference, the ISP may want to add the costs of computational resources consumed by the H.323 proxy. These costs may become significant if, for example, the H.323 proxy also provide content clipping to the benefit of resource-constrained and bandwidth-limited client users (e.g., wireless personal digital assistants);

4) Finally, every 10 seconds the accounting system must communicate qualifying accounting data records from step 1, 2, and 3 to the ISP's billing and invoice services. In turn, these will provide the actual cumulative cost figure in real-time to User A.

The accounting system that emerges from this description is beyond the state of the art. The AIACE architecture makes some fundamental contributions to the realization of this environment. AIACE's contributions derive from locating an accounting co-processor within the firewall, and from populating it with several specialized plug-ins.

A plug-in running in the co-processor capitalizes on the proximity with the H.323 proxy. From the H.323 proxy, this plug-in—the H.323 correlation plug-in—derives the information needed to quickly correlate traditional L3/L4-based accounting data records into a unique accounting record for the whole H.323 session. Another plug-in running in the co-processor monitors the consumption of computational resources by the H.323 proxy. This plug-in—the CPU cycles plug-in—extends traditional accounting data to include information regarding CPU cycles, scheduling events, etc. that were caused by the H.323 proxy.

[4] L3, L4, and L7 refer to layers in the OSI reference model.

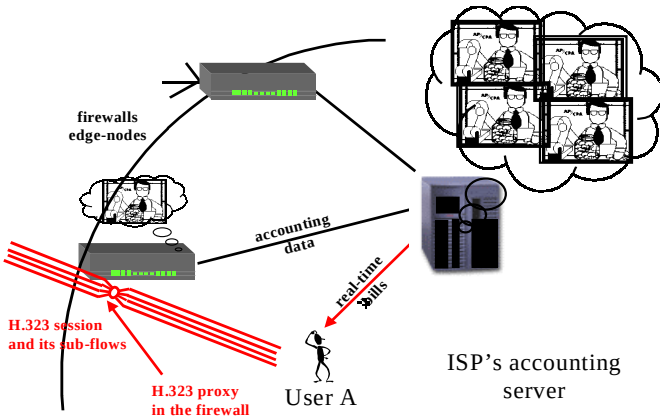


Fig. 6. IP flows belonging to the same H.323 session are aggregated at the network node.

Another plug-in running in the co-processor—the real-time transfer plug-in—expedites the transfer of the H.323-related accounting data to the accounting/billing server(s). This way, the accounting/billing servers can meet the real-time customer care requirements.

Let us further assume that User A uses a pre-paid debit card to secure funds to the ISP. In this case, an additional accounting plug-in—the pre-paid debit card plug-in— joins the co-processor and decrements User A's credit until there are no credits. When the credit hits zero, this plug-in terminates the H.323 proxy and start reclaiming the accounting resources related to User A's H.323 session.

Similar pre-aggregation opportunities at the network nodes arise when different logical groups of resources are being considered. VLANs, for instance, are a L2 paradigm to group stations on different LAN segments based on explicit IEEE 802.1Q tags, or IP addresses, or protocol numbers, or ingress/egress interfaces. In all cases but the latter, the aggregation of accounting data on a per-VLAN basis is not trivial. The AIACE co-processor can efficiently pre-aggregate this accounting data using network-node local knowledge (e.g., IEEE 802.1Q tags).

B. High-confidence traffic monitoring

A high-confidence traffic monitoring application periodically sweeps a network topology and reports network vitals to the operator (e.g., the cumulative traffic figures concerning the most remunerative SLAs). The high-confidence attribute implies that such an application is dependable in reporting traffic figures in real-time, in spite of overloads (e.g., network, CPU, or accounting data overloads) possibly induced by partial failures in the network. In other words, this type of traffic monitoring application must be especially well behaved when things in the network start to

go wrong. Whenever applicable, the terminology is being deliberately borrowed from mission-critical real-time applications such as air traffic control ([6]).

In this application, it is crucial to manage the finite monitoring capacity and make the most effective accounting data mining out of it. A whole sweep of the network topology represents a cycle; cycles are typically configured to complete in a handful of seconds. At the end of each cycle, the breakdown of the monitoring capacity is revisited to adapt to conditions occurred in the previous sweep, or to accommodate an operator's explicit request to zoom-in on "hot" sectors of the sweep. In principle, at each cycle the traffic monitoring application communicates to AIACE network nodes:

1) How much accounting data the application wants to handle from a given network node;

2) How should the network node weight its accounting data, decide what to mine out, and package it within the aforementioned limit;

3) Which accuracy is expected from the network node while performing this accounting data mining.

It is appropriate to think of Steps 1 to 3 as a generalization of RMON's Host Top N group variables. The opportunities in Step 2 become evident in the following example (Fig. 7). The network under analysis is QoS-enabled and three QoS classes—gold, silver, and bronze—are defined. By specifying a weighting algorithm for accounting data in the various QoS classes, the application passes tidbits of its business model to the network node—i.e., it says what the most significant accounting data are and how much this matters. The network node can use the weighting algorithm to rank the accounting data that best reflect this business model. Should the accounting data exceed the size pre-set by the application for a given node (overload), that node will throttle itself by pruning the least significant accounting data from its reports.

AIACE's accounting plug-ins realize Steps 1 through 3 by operating both at the network nodes and at the accounting server. The plug-ins that define the weighting algorithms load and execute at the network node. The plug-ins that implement the accounting wire protocol and its capability to drive the network node's accounting data output are reciprocated at the network node and at the accounting server.

VI. Related Work

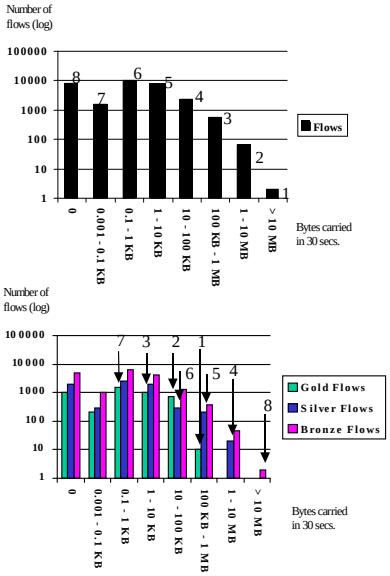
Aboba and Jarkko [1] document several key definitions and taxonomies for accounting and billing. They also survey existing practices for accounting and billing. In this paper, AIACE's scope has been limited to what [1] refers to as the accounting protocol between a network node and the accounting server. Inter-domain and intra-domain protocols among accounting servers are left as an unexplored application area for AIACE's accounting plug-ins.

The meters and flow analysis in OC3MON [2] are configurable and programmable; new native code can be added to produce custom delay lines or replay behaviors. AIACE plays at a higher level of abstraction. While it is possible to realize delay lines or replay with AIACE plug-ins, AIACE's emphasis is on making network nodes communicate intelligently with applications, and have applications delegate activities such as pre-paid debit card semantics to network nodes.

Pricing strategies for data services (as in [17]) are orthogonal to AIACE's thrust (i.e., policy vs. mechanism). Adaptive, usage-based pricing as in [17], however, supports and sustains AIACE's motivations.

With Active Networks [21], researchers have blurred the dichotomy between transient data packets and node-resident protocols. Both the integrated Active Network approach—the code is carried by the packet "by value"—and the discrete Active Network approach—the code is carried by the packet "by reference"—have been described in literature.

One of the first accounts of Active Networks themes being applied to accounting and billing can be found in [20].



a) In this example, a network-node organizes about 2 million PDU traces into 30,000 IP flows. It classifies the resulting flows based on the bytes transferred on each flow. It then ranks flows (from 1 to 8). The higher the rank number, the higher the chance that the flow will not be transferred to the accounting server in case of data overload.

b) The node now structures the same accounting data into QoS-flavored flows (same X and Y axis as in a). After applying a QoS-specific weighting algorithm to the flows, the node ranks flows with different results than a). The weighting algorithm can be arbitrarily complex and take into account other considerations besides bytes transferred (e.g., hosts, number of packets, duration).

Fig. 7. A network node ranks accounting data through different weighting algorithms in a) and b). Should overload occur, the node will omit the least significant (highest-ranking) accounting data from its real-time reports to the accounting server.

With self-paying transport, PDUs are extended with "hand-carried" electronic payment information. AIACE applies Active Networks themes to the accounting co-processor. AIACE, however, is limited to the management plane, it does not interfere with the end-to-end path, and thus it may find paths of lesser resistance towards adoption.

Likewise, AIACE can be regarded as an original application of Liquid Software [8] to a path that is not the end-to-end path, but is rather the IP accounting and management path.

In [13], the accounting/billing infrastructure meets Active Networks with a radical new approach: the accounting and billing functions are delegated to customer's systems. AIACE shares the same fundamental objectives, i.e. a highly customizable service to the end user. In AIACE, however, this objective is reached through an effective processing of accounting data. Unlike [13], AIACE does not depend upon customer systems nor is it exposed to their misbehaving.

The problems of using Java within a shared environment such as a network node have been explored in [5]. These problems can lead to denial of service due to resource theft by plug-ins that maliciously or inadvertently consume disproportionate amounts of resources. That work has produced a resource-safe Java platform that is ideal to realize the AIACE co-processor and its guaranteed operations.

VII. Conclusions

AIACE is an active IP accounting infrastructure for next-generation accounting requirements that arise from Internet economics, voice and data integration, and new applications (such as real-time billing). Through AIACE's node-resident accounting co-processor, specialized accounting plug-ins can work in place at the network node, where the accounting data is being generated and packaged. Several effectiveness and efficiency scenarios support the vision that AIACE's flexibility will benefit service providers, network operators, and ultimately the end users.

In the long term, AIACE's most distinguishing contributions are expected in those scenarios where accounting applications have a rich context and accounting data mining has to take place in real-time, with sub-minute latencies. With the accounting node co-processor, it is finally possible to marry Active Networks themes with real-time solutions to the management of accounting data and overloads; these solutions are inspired by results in real-time distributed systems and mission-critical applications [6].

Acknowledgment

The author is indebted to many colleagues at the Nortel Networks Technology Center for stimulating discussions on Active Networks. The author is particularly grateful to Adnan Onart for providing invaluable feedback on the paper. The results of the work by Vikas Arora, Rob Duncan, Rob Jaeger, and Tal Lavian was of great encouragement while writing this paper. The author is grateful to the anonymous reviewers whose comments have greatly improved the description of AIACE.

References

[1] B. Aboba, J. Akko, "Introduction to Accounting Management," Internet Draft, draft-aboba-acct-01.txt.
[2] J. Apisdorf, K. Claffy, K. Thompson, and R. Wilder, "OC3MON: flexible, affordable, high performance statistics collection," http://www.nlanr.net/NA/Oc3mon.
[3] V. Arora, et al. "An IP Accounting Server (IPAS)," unpublished Nortel Networks TR.
[4] A. Begel et al., "BPF+: Exploiting Global Data-flow Optimization in a Generalized Packet Filter Architecture," SIGCOMM'99, Cambridge, MA, Sep. 99.
[5] P.Bernadat, D. Lambright, and F. Travostino, "Towards a Resource-safe Java for Service-Guarantees in Uncooperative Environments," *PLRTIA '98*, Madrid, Spain, Dec. '98.
[6] R. Clark, et al. "An Adaptive, Distributed Airborne Tracking System," Workshop on Parallel and Distributed Real-Time Systems, San Juan, PR, April 99.
[7] H.323 standard, http://www.itu.int.
[8] J. Hartman, et al. "Liquid Software: A new Paradigm for Networked Systems," Technical Report 96-11, Dept. of Computer Science, The University of Arizona.
[9] R.Jaeger, T.Lavian, "Dynamic Classification in Silicon Based Forwarding Engine Environments", Proceedings of LANMAN'99 Workshop, Sidney, Nov. 1999.
[10] S.O.Malley and L.L.Peterson, "A Dynamic Network Architecture," ACM Transactions on Computer Systems, (10)2, May 1992.
[11] NetFlow,http://www.cisco.com/warp/public/cc/cisco/mkt/ios/netflow/tech/napps_wp.htm.
[12] Optivity Service Accounting (OSA), Nortel Networks.
[13] M. Rizzo, et al. "A Dynamic Pricing Framework to Support a Scalable, Usage-Based Charging Model for Packet-Switched Networks", IWAN '99, Berlin, Germany. May 24-26 1999.
[14] RADIUS Accounting, IETF RFC 2139.
[15] RMON MIB, IETF RFC 1757.
[16] RMON MIB extensions for Switched Networks, IETF RFC 2613.
[17] S. Shenker, et al., "Pricing in Computer Networks: Reshaping the Research Agenda," Xerox Parc TR.
[18] SIP, IETF RFC 2543
[19] SNMP, IETF RFC 1157.
[20] J.M. Smith et al., "SwitchWare: Towards a 21st Century Network Infrastructure," University of Pennsylvania TR.
[21] D. L. Tennenhouse and D. Wetherall. "Towards an active network architecture." In Multimedia Computing and Networking 96, San Jose, CA, Jan 1996.