Quality of Service in Wireless Networks using Weighted Clustering Algorithm

M. Anil Kumar[1], V. Srikanth[2], N. Vijaya Gopal[3]

[1]K L University, Department of CSE, Vaddeswaram, Guntur 522502, Andhra Pradesh, India
modhugula.anilkumar@hotmail.com

[2]K L University, head of the department CSE, Vaddeswaram, Guntur 522502, Andhra Pradesh, India
vsrikanth@kluniversity.in

[3]K L University, Department of CSE, Vaddeswaram, Guntur 522502, Andhra Pradesh, India
nvgopal63@hotmail.com

Abstract: ***In today's Internet, inter-domain route control remains indefinable; however, such control could improve the performance, dependability, and usefulness of the network for end users and ISPs alike. While researchers have anticipated a number of resource routing techniques to combat this limit, there has thus far been no way for autonomous AS to ensure that such traffic does not circumvent local traffic policies, nor to exactly determine the correct party to charge for forwarding the traffic.***

Keywords: Authentication, capabilities, overlay networks, source routing

1. Introduction

Network operators and academic researchers alike recognize that today's wide-area Internet routing does not realize the full potential of the existing network infrastructure in terms of performance, reliability, or flexibility. While a number of techniques for intelligent, source-controlled path selection have been proposed to improve end-to-end performance, reliability, and flexibility, they have proven problematic to deploy due to concerns about security and network instability. We attempt to address these issues in developing a scalable, authenticated, policy-compliant, wide-area source routing protocol.

We argue that many of the deficiencies of today's routing infrastructure are symptoms of the coupling of routing policy and routing mechanism. In particular, today's primary wide area routing protocol, the Border Gateway Protocol (BGP), is extraordinarily difficult to describe, analyze, or manage. Autonomous systems (AS) express their local routing policy during BGP route advertisement by affecting the routes that are chosen and exported to neighbors. Similarly, AS often adjust a number of attributes on routes they accept from their neighbors according to local guidelines. As a result, configuring BGP becomes an overly complex task, one for which the outcome is rarely certain. BGP's complexity affects Internet Service Providers (ISPs) and end users alike; ISPs struggle to understand and configure their networks while end users are left to wonder why end-to-end connectivity is so poor.

We present the design and evaluation of Platypus, a source routing system that, like many source-routing protocols before it, can be used to implement efficient overlay forwarding, select among multiple ingress/egress routers, provide virtual AS multi-homing, and address many other common routing deficiencies. The key advantage of Platypus is its ability to ensure policy compliance during packet forwarding. Platypus enables packets to be stamped at the source as being policy compliant, reducing policy enforcement to stamp verification. Hence, Platypus allows for management of routing policy independent of route export and path selection.

2. System Analysis

The first step in developing anything is to state the requirements. This applies just as much to leading edge research as to simple programs and to personal programs, as well as to large team efforts. Being vague about your objective only postpones decisions to a later stage where changes are much more costly.

The problem statement should state what is to be done and not how it is to be done. It should be a statement of needs, not a proposal for a solution. A user manual for the desired system is a good problem statement. The requestor should indicate which features are mandatory and which are optional, to avoid overly constraining design decisions. The requestor should avoid describing system internals, as this restricts implementation flexibility. Performance specifications and protocols for interaction with external systems are legitimate requirements. Software engineering standards, such as modular construction, design for testability, and provision for future extensions, are also proper.

Problem statements range from individuals, companies, and government agencies, mixture requirements with design decisions. There may sometimes be a compelling reason to require a particular computer or language; there is rarely justification to specify the use of a particular algorithm. The analyst must separate the true requirements from design and implementation decisions disguised as requirements. The analyst should challenge such pseudo requirements, as they restrict flexibility. There may be politics or organizational reasons for the pseudo requirements, but at least the analyst should recognize that

these externally imposed design decisions are not essential features of the problem domain.

A problem statement may have more or less detail. A requirement for a conventional product, such as a payroll program or a billing system, may have considerable detail. A requirement for a research effort in a new area may lack many details, but presumably the research has some objective, which should be clearly stated.

Most problem statements are ambiguous, incomplete, or even inconsistent. Some requirements are just plain wrong. Some requirements, although precisely stated, have unpleasant consequences on the system behavior or impose unreasonable implementation costs. Some requirements seem reasonable at first but do not work out as well as the request or thought. The problem statement is just a starting point for understanding the problem, not an immutable document. The purpose of the subsequent analysis is to fully understand the problem and its implications. There is no reasons to expect that a problem statement prepared without a fully analysis will be correct.

The analyst must work with the requestor to refine the requirements so they represent the requestor's true intent. This involves challenging the requirements and probing for missing information. The psychological, organizational, and political considerations of doing this are beyond the scope of this book, except for the following piece of advice: If you do exactly what the customer asked for, but the result does not meet the customer's real needs, you will probably be blamed anyway.

3. Modules

1. Networking Module.
2. ISP Module.
3. Load Balancing Module.
4. Platypus Framework Module.

1. Networking Module:

Client-server computing or networking is a distributed application architecture that partitions tasks or workloads between service providers (servers) and service requesters, called clients. Often clients and servers operate over a computer network on separate hardware. A server machine is a high-performance host that is running one or more server programs which share its resources with clients. A client also shares any of its resources; Clients therefore initiate communication sessions with servers which await (listen to) incoming requests.

2. ISP Module:

Autonomous systems (AS) express their local routing policy during BGP route advertisement by affecting the routes that are chosen and exported to neighbors. Similarly, AS often adjust a number of attributes on routes they accept from their neighbors according to local guidelines. As a result, configuring BGP becomes an overly complex task, one for which the outcome is rarely certain. BGP's complexity affects Internet Service Providers (ISPs) and end users alike; ISPs struggle to understand and configure their networks while end users are left to wonder why end-to-end connectivity is so poor.

3. Load Balancing Module:

We consider a policy in which the server selects a set of waypoints to forward traffic through and load balances across them. This functionality is important in many applications, since it is unlikely that a single waypoint can suffice for an arbitrarily large traffic volume.

We evaluate a Web server application scenario with probabilistic load balancing across two waypoints. Each client makes ordinary HTTP requests to the server. The server's replies are stamped according to a policy that begins by sending all response traffic through a single waypoint. Halfway through the experiment we change the policy such that the response traffic is load balanced at the granularity of a TCP flow.

4. Platypus Framework Module:

We detail the design and implementation of a policy framework for managing Platypus in an AS. Incremental deployability is key in our setting, as it would be unreasonable to expect AS to cooperate in the deployment of a system that affects local policy.

Our policy framework in an in-network stamping scenario that uses DNS-based delegation. Central to the framework is the Policy Engine, which implements the AS's policy. The policy engine instructs the stamper and a DNS component based on policy, and obtains delegated capabilities through them. The stamper and policy engine are implemented inside a Platypus router, which is located on the path of inbound and outbound traffic.

4. Algorithm /Method Used

Platypus Policy Framework

Algorithm /Method Description:

Platypus uses network capabilities, primitives that are placed within individual packets, to securely attest to the policy compliance of source routing requests.

Network capabilities are;

i. Transferable: an entity can delegate capabilities to others,

ii. Composable: a packet may be accompanied by a set of capabilities

iii. Cryptographically authenticated. Capabilities can be issued by AS to any parties they know how to bill. Each capability specifies a desired transit point (called a waypoint), a resource principal responsible for the traffic, and a stamp of authorization.

5. Requirements

SDLC Methodology:

This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, "A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

- The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
- A preliminary design is created for the new system.
- A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
- A second prototype is evolved by a fourfold procedure:

    1) Evaluating the first prototype in terms of its strengths, weakness, and risks.
    2) Defining the requirements of the second prototype.
    3) Planning a designing the second prototype.
    4) Constructing and testing the second prototype.

- At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer's judgment, result in a less-than-satisfactory final product.
- The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
- The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
- The final system is constructed, based on the refined prototype.
- The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

The following diagram shows how a spiral model acts like:

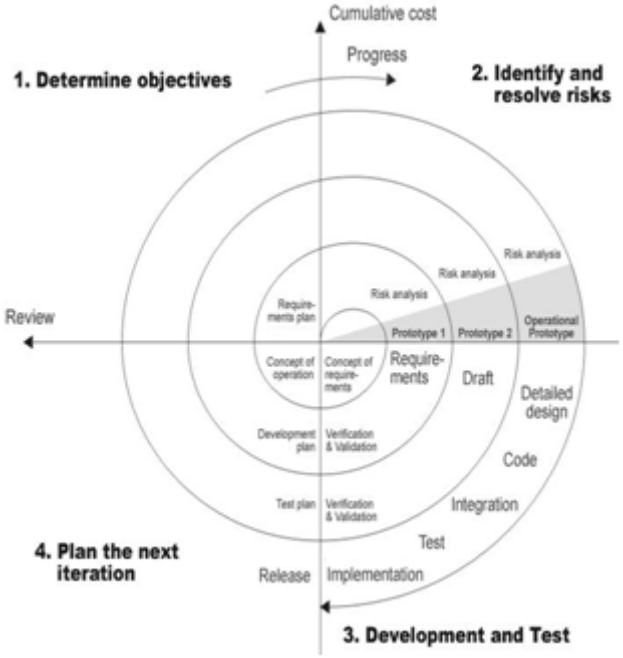

Figure 1: Spiral Model

Advantages:

- Estimates (i.e. budget, schedule etc.) become more realistic as work progresses, because important issues discovered earlier.
- It is more able to cope with the changes that are software development generally entails.
- Software engineers can get their hands in and start working on the core of a project earlier.

6. Conclusion

We argue that capability is individually well-suited for use in wide-area Internet routing. The Internet serves an exceedingly large number of users with an even larger number of motivations, all attempting to concurrently share widely distributed resources. Most significantly, there exists no single authority that can make informed access decisions. Also, we believe that much of the complexity of Internet routing policy stems from in flexibility of existing routing protocols. We want to study how one might implement inter - AS traffic engineering policies through capability price strategies. For example, an AS with multiple peering routers that wishes to support load balancing may be able to do so from end to end variable pricing of capabilities for the equivalent Platypus waypoints. While appropriately modeling the self-regarding behavior of external entities may be difficult, we are confident that this challenge is simplified by the direct mapping between Platypus waypoints and path selection.

References

[1] S. Agarwal, C.-N. Chuah, and R. H. Katz, "OPCA: Robust inter domain policy routing and traffic control," in Proc. IEEE OPENARCH, Apr. 2003, pp. 55–64.
[2] M. K. Aguilera, M. Ji, M. Lillibridge, J. MacCormick, E. Oertli, D. G. Andersen, M. Burrows, T. Mann, and C. A. Thekkath, "Block-level security for network-attached disks," in Proc. USENIX FAST, Apr. 2003.
[3] D. G. Andersen, "Mayday: Distributed filtering for Internet services," in Proc. USITS, Mar. 2003.
[4] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. T. Morris, "Resilient overlay networks," in Proc. ACM SOSP, Oct. 2001.
[5] R. Atkinson, "Security architecture for the Internet protocol," in IETF, RFC 1825, Aug. 1995.
[6] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz, "The effects of asymmetry on TCP performance," in Proc. ACM Mobicom, Sep. 1997.
[7] M. Bellare, R. Canetti, and H. Krawczyk, "Pseudorandom functions revisited: the cascade construction and its concrete security," in Proc. IEEE FOCS, 1996, pp. 514–523