University of Pennsylvania
ScholarlyCommons

Departmental Papers (CIS)

Department of Computer & Information Science

February 2003

Application-Private Networks

Jonathan M. Smith

University of Pennsylvania, jms@cis.upenn.edu

Follow this and additional works at: https://repository.upenn.edu/cis_papers

Recommended Citation

Jonathan M. Smith, "Application-Private Networks", . February 2003.

Postprint version. Published in *Computer Systems: Theory, Technology and Applications: A Tribute to Roger Needham*, New York : Springer, 2004, pages 273-277. Proceedings of the meeting, 'Roger Needham: 50 and 5', held 17 February 2003.

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/cis_papers/175
For more information, please contact repository@pobox.upenn.edu.

Application-Private Networks

Abstract

The design space for network architectures can be conveniently described as a 3-tuple of $\langle \text{Application Requirements, Protocol Elements, Network Conditions} \rangle$. *Application requirements* can range from reliability and small message inter-arrival delay to communications secrecy. *Protocol elements* include acknowledgements and error-correcting codes, timers and a variety of cryptographic transformations. *Network conditions* include delay, delay variance, loss rates, bit error rates (BERs), topology, and available bandwidths. For any given triple, and in particular for a choice of application and requirements, there are assumptions about operating conditions made, and protocol elements selected to meet the application requirements under these conditions.

Comments

Postprint version. Published in *Computer Systems: Theory, Technology and Applications: A Tribute to Roger Needham*, New York : Springer, 2004, pages 273-277. Proceedings of the meeting, 'Roger Needham: 50 and 5', held 17 February 2003.

Application-Private Networks

Jonathan M. Smith

University of Pennsylvania, Philadelphia, Pennsylvania, USA

Introduction

The design space for network architectures can be conveniently described as a 3-tuple of *<Application Requirements, Protocol Elements, Network Conditions>*. *Application requirements* can range from reliability and small message inter-arrival delay to communications secrecy. *Protocol elements* include acknowledgements and error-correcting codes, timers and a variety of cryptographic transformations. *Network conditions* include delay, delay variance, loss rates, bit error rates (BERs), topology, and available bandwidths. For any given triple, and in particular for a choice of application and requirements, there are assumptions about operating conditions made, and protocol elements selected to meet the application requirements under these conditions.

Two examples, the telephone network and the Internet, are useful in understanding this architectural framework. The telephone network in its purest form is engineered [1] to deliver a band-limited audio channel appropriate for interactive voice telecommunications. The application requirements then, include the ability to deliver about 3000Hz of audio, with some limits on delay and audible impairments. These requirements have been met in the telephony architecture by using a call set-up protocol of considerable complexity to establish a point-to-point channel for carrying a voice stream. Link, multiplexing, switching and capacity engineering are voice-centric.

The Internet design, requiring interoperation across a variety of networks and operating conditions, and intended to service many applications, must choose protocols that can tolerate an extremely wide variety of network conditions. Thus, the basic IP transport service is a minimal datagram service, response to network dynamics such as topology changes is provided by dynamic routing and other application requirements (ordering, reliability, etc.) are provided by end-to-end overlay protocols such as the Transmission Control Protocol, TCP.

If we contrast the Internet architecture with the telephony network architecture, TCP/IP is intended to be agnostic with respect to applications, and adapts to a large (but not all-encompassing) range of network conditions with its choice of protocol elements. To optimize the placement of protocol functions in the architecture (as opposed to for a specific requirement) the “end-to-end” design notion pushes functions to the end-points, eliminating redundant implementation and giving application designers the widest range of options for use of the basic network service.

These two examples illustrate the design space and tradeoffs made amongst its “dimensions.” Neither architecture is ideal — for example the attempt to remove many dynamics in network conditions within the call makes the telephony architecture limited in its ability to efficiently handle applications with dynamics very different than that of voice. Likewise, the IP architecture’s engineering approach to dealing with many

applications and network conditions has forced engineering tradeoffs, such as substantial over-provisioning (to control delay jitter) to support applications such as voice and video.

Automated optimal network engineering

An ideal network architecture, within the constraints of our design space, would have the property that at any given time, the application requirements and network conditions would result in the best known selection and placement of protocol elements. For example, if network condition dynamics result in a variable BER, as in a mobile wireless context, the protocol architecture might be adjusted to inject forward error correction (FEC) to move TCP/IP into an operating regime where its protocol element selections result in meeting application requirements. While limited instances of such techniques have been demonstrated experimentally [4], the ideal system would *automate* [6] such responses, under control of high-level models of application requirements.

A great deal of detail is masked by the design space abstraction presented in the Introduction but the basic point is not to be lost: for any specified application requirements (including preferences, weights, *etc.*) and network conditions (we will discuss how information about such network conditions might be made available using the “Knowledge Plane” proposed by David Clark [3], in the next section), one or more equivalent selections of protocol elements can be made which closely meet the application requirements. As this process is fundamentally driven by application requirements, we call such networks *Application-Private Networks*, or *APNets*. The basic design process for an *APNet*, for a particular application, would result in a protocol architecture optimized for that application’s performance, with protocol elements selected in concert with any techniques, such as time-division multiplexing, needed to limit the range of network conditions for these selections. The resulting network architecture is colloquially called a “stovepipe.”

An excellent example design from the space systems domain is the “Remote Agent” [6] architecture used in NASA’s *Deep Space One (DS1)* mission, where many of the challenges are similar to those of network engineering, such as multiple timescales, unplanned events, and overall “mission goals.” In the NASA system, very high-level models are used to drive a planning system; current conditions are fed into a system with a limited time horizon to drive specific actions such as recovery, reconfiguration and reprogramming in the face of system conditions such as failed sensors and actuators.

The challenge in the more general case is large-scale sharing. That is, “stovepipe” design is economically inefficient, inhibits adaptation and reuse, and makes interoperability with other applications, as well as sharing of facilities, difficult. Further, it makes unfounded assumptions for the general case, where conflicting goals between users are common. The advent of programmability in many network components, such as network processors, software radios and extensible routers, permits the configuration of such components to be *virtualized*. That is, the component behaviour can support multiple application-driven specializations. The problem is not easy, but is conceptually within reach [6], as demonstrated by the DS1 experiments we have discussed. An abstraction is given in Figure 1a, where application requirements (specified, perhaps as in the next section) induce behaviors at various logical levels in a network, from host to link.

This process will take place repeatedly based on changes in network conditions. The reconfiguration process must be safe, network knowledge must be available to both the protocol element selection and programmable component configuration processes, and the network knowledge must be trusted, to deal with accidental and malicious failures.

Figure 1a: APNet Configuration

Figure 1b: Controlling APNet Dynamics

Among the interesting technical questions to be resolved are issues of security, stability and degree of extensibility for the architecture as a whole. To touch just briefly on these issues, the degrees of extensibility might include those possible from a machine learning algorithm in optimization of protocol selections, they might include addition of new protocol elements as they are discovered, or they might include wholesale changes of the control architecture itself. Stability issues include overreactions, damping and convergence of distributed control schemes. Prototyping and experiments can identify the appropriate adaptation rates for various timescales, ranging from the immediate to relatively long-term, which some researchers have categorized as *reactive*, *deliberative* and *reflective* — Figure 1b illustrates how these adaptation timescales might affect the dynamics of *APNet* instances. Security concerns, in addition to the trust of network condition data, include the risk of subtle Denial of Service attacks on a complex infrastructure, data privacy, authorization for code loading, provenance of aggregated data, and finally, the technically difficult issue of what the telephony industry politely refers to as “feature interaction.”

Trust architecture for network knowledge

The interaction between the “Knowledge Plane” and *APNets* is important, and if network knowledge is to be widely used it will be named. Much knowledge will be represented syntactically as strings of the form `<name>=<value>`, e.g., “bandwidth=64K.” This scheme has been widely adopted, in contexts from scripting languages to WWW “cookies,” and is readily translated to locally convenient representations. An example use of such a variable is the `TERM` variable used to configure terminal handling in some operating systems in concert with a database of information about terminal capabilities. In

an *APNet*, the host operating system might, using the variables specified by the application, configure schedulers, networking stacks, and choose network adapters.

The string representation enables use of *Trust Management* [2] technology such as the KeyNote [5] system, which represents *assertions* as credentials with *authorizers*, *licensees* and *conditions*. Public-key technologies are used to build the web of trust, and a compliance checking process is used to test requested actions against the credentials. Consider public keys for *rmn* and *jms77*, where *jms77*'s key is the licensee, *rmn*'s key is the authorizer, conditions are

```
$file_owner="rmn" && $filename="/home/rmn/[^/]*"  
&& $hostname = "ouse.cl.cam.ac.uk" -> "true"
```

and the signature is with *rmn*'s key. Then *jms77* is authorized by *rmn* to access files in *rmn*'s home directory on a particular host at the University of Cambridge.

This architecture provides capability-like [7] control of resources and robust delegation of authority in spite of distributed control through its use of cryptography to authenticate and authorize remote operations [8], and has many other desirable features. Complete explication would more space, but among the desirable properties of credentials and a trusted knowledge plane for advanced applications are: data provenance, support for micro-payment systems of various flavors, authorization for network control, code-loading, resource allocation and digital-rights management.

Conclusion

Application-Private Networks extend the range of dynamics for protocol architectures, by dynamically selecting protocol elements to meet application requirements in the face of dynamic conditions. Such a network architecture is not only desirable, it is technically achievable within the next decade. A broad range of new network uses are enabled.

References

1. **Engineering and operations in the Bell System** (2nd ed.), AT&T Bell Laboratories, Murray Hill, NJ, 1983 (ISBN #0-932764-04-5).
2. BLAZE, M., FEIGENBAUM, J. AND LACY, J., 'Decentralized trust management,' *Proc. IEEE 17th Symposium on Security and Privacy*, 1996, pp. 164-173.
3. CLARK, D., "A new vision for network architecture," *private communication*, September 2002.
4. HADZIC, I., 'Applying reconfigurable computing to reconfigurable networks,' *Ph.D. Thesis*, Department of Electrical Engineering, University of Pennsylvania, 1999.
5. KEROMYTIS, A., ET AL., 'The STRONGMAN architecture,' to appear in *3rd DARPA Information Survivability Conference and Exposition (DISCEX)*, April 2003.
6. MUSCETTOLA, N., NAYAK, P., PELL B. AND WILLIAMS, B. C., 'Remote agent: to boldly go where no AI System has gone before,' *Artificial Intelligence*, vol. 103, no.1-2, pp. 5-48.

7. NEEDHAM, R.M. AND WALKER, R. D. H., 'The Cambridge CAP computer and its protection system,' in *Proc. 6th Symposium on Operating Systems Principles*, Nov. 1977, pp. 1-10.
8. NEEDHAM R.M. AND SCHROEDER, M.D., 'Using encryption for authentication in large networks,' *Comm. ACM*, vol. 21, no.12, 1978, pp. 993-999.