Section 2: Week 3: Communication Mechanisms

Nate Bachmeier

TIM-8120: Distributed Systems

October 6th, 2019

North Central University

**Communication Mechanisms**

Distributed systems are composed of systems that need to participate in conversations across the network. The requirements of these conversations will as the participants optimize network overhead, service response time, higher hit ratios, and scalability considerations (Chen & Sung, 2016). Due to applications having different constraints, there is an assortment of protocols that make trade-offs in terms of scale or performance. Chen and Sung, provide the example of Microsoft’s Universal Plug and Play (UPnP), which can easily share services on a local network but cannot scale to an enormous enterprise environment. These choices naturally led to systems designers needing to be cognisent of the target audience when selecting a network protocol.

# Common Protocols in Networked Applications

The Open Systems Interconnection (OSI) model represents network communication as a series of layers (Zimmermann, 1980). Implementation of each layer can focus on the immediate task at hand, as there are clear sepearation of duties with other aspects of the system.

|  |  |  |
| --- | --- | --- |
| OSI Level | Purpose | Examples |
| Application  (Layer 7) | Provides the interface between the software running on the device and the network protocol | Secure Shell (SSH): Remote Administration  File Transfer Protocol (FTP): File Administration  Domain Name Services (DNS): A protocol for translating network names into IP addresses.  Bonjour: Protocol for name translation on OSX |
| Presentation  (Layer 6) | Responsible for packet forwarding and routing through intermediate routers | HyperText Transport (HTTP): Stateless protocol for request/response traffic  Simple Network Management Protocol (SNMP): Stateless protocol for communicating with switches |
| Session  (Layer 5) | Provides process communications between two or more networked devices | Remote Procedure Calls (RPC): Share application state across disjoined systems  Named Pipes: An Inter-Process Communication (IPC) mechanism  Server Message Block (SMB): Windows-to-Windows protocol of choice for administration scenarios. |
| Transport  (Layer 4) | Transfers data between various systems and hosts | Transmission Control Protocol (TCP): Mechanism for ensuring delivery and resending packets within a stream  User Datagram Protocol (UDP): A lightweight system for best effort delivery and multi-casting scenarios  Border Gateway Protocol (BGP): Defines network edge to edge routes. |
| Network Link (Layer 3) | Provides switching and routing functionality between connected devices | Internet Protocol (IP): A route addressing solution for getting packets across the Internet.  Virtual Private Network (VPN): Encapsulates traffic and then routes it to a different network. |
| Data Link  (Layer 2) | A link between two directly connected nodes. Detects and corrects errors transmitted from the physical layer | Ethernet: A technology for detecting and correcting failures on the wire.  Address Resolution Protocol (ARP): Translates IP-to-MAC addresses between switches. |
| Physical Layer (Layer 1) | Defines the connection between the physical device and its connection mechanism | CAT-5: An encoding scheme for reading the electrical signal.  Universal Serial Bus (USB): Connects devices through a generic interface.  802.11 (Wi-Fi): Radio communication frequently used in offices, coffee shops, and other wireless scenarios. |

# Network Management Suites

Distributed systems can reduce their net costs, through higher utilization by sharing storage and compute resources across multiple tenants (González-Férez & Bilas, 2016). These resource allocations become multi-process decompositions, as the High Performance Computing Service (HPCS) breaks them into manageable units of work (Thiele, et al.). For example, YouTube subscribers upload 500 hours of video every single minute (Celement, 2019). These videos need to be encoded, replicated, and indexed across network systems. It would be challenging for a traditional computer operating system to manage that process, as their scope of control is too *centralized* and optimized for organizing *local* resources. Network Management Suites can address the *decentralized* nature of these problems to organize *remote* resources.

|  |  |  |
| --- | --- | --- |
| Product | Pros | Cons |
| Apache Hadoop (Apache Hadoop, 2019) | General purpose solution for distributed file management | Complex to provision and manage. Major cloud providers have addressed this challenge with managed solutions.  It requires an ecosystem or related tools for many use cases. |
| Apache Yarn  (Apache, 2019) | General purpose task orchestration. | Requires specialized frameworks and programming models to efficiently leverage. |
| Kubernetes (Google, 2019) | Cloud agnostic container orchestration removes the need for homogeneous deployments. | Pushes designers toward the least common denominator as the Cloud Platform Native features are masked |
| Microsoft AppFabric | A Single System Compute environment that hides many distributed system concepts. | Requires code follows specific patterns (Stateful Providers and Stateless Providers). Targets the .NET ecosystem |
| Amazon EFS (Amazon, 2019) | Projects a cluster of storage resources as a single Network File System (NFS) endpoint | Administrators need to mount the remote resources, as local storage. High risk of network latency. |
| IBM Service Management Suite (IBM, 2019) | Fully manage an IBM centric environment. | It only works with IBM servers. |
| Docker Swarm (Docker, 2019) | Open source cluster management of container orchestration. | Focuses on scheduling containers, the administrator still needs to provision resources and configure storage out of bound. |
| Redhat Satallite (Redhat, 2019) | Automates provisioning and management of collections of Redhat instances. | It requires experts with in-depth technical knowledge of the product space. |
| Microsoft System Center (Microsoft, 2019) | Holistic management of Hybrid Cloud Enterprise Environments. | It focuses on large environments that need that manage thousands of nodes. |

# Network Programming Languages

There are multiple programming languages and frameworks for creating networked applications.

|  |  |
| --- | --- |
| Language | Features |
| C++ | The cross language of choice, when performance and portability are the core concerns. |
| Java/C# | Type-safe object oriented languages with vast frameworks and documented examples. |
| Node/JavaScript | An event driven programming language that is light weight and uses a minimalistic approach to resources. |
| Erlang (Ericsson, 2019) | A functional language that has built-in capabilities for fault tolerance and concurrency management. |
| Rust (Klabnik & Nichols, 2018) | A functional language that compiles into native assemblies. Unlike C, there are several rules to make the language safe from many memory corruption scenarios. |
| Python | A general purpose functional/object centric language that leverages vast collections of open source modules for rapid development. |
| Swift/Object-C | These are client side languages that simplify the development of iOS based mobile applications. |

# Expanding the Systems Diagram