Thinking about networks

Ankit Singla

Small number of slides adapted from Scott Shenker, Jennifer Rexford, Laurent Vanbever
Learning objectives

• What do we want from the network?
• How do we achieve these goals?

• Readings for next week
  • Reading 1: Rethinking the design of the Internet
  • Reading 2: Why the Internet only just works
What do we want from the network?

• Performance: latency? bandwidth?
• Reliability / availability / security?
• Flexibility, manageability?
• …
Network performance
Performance = bandwidth?
To get many megabits-per-second ...
To get many megabits-per-second ...

4400 km
80 Km / hour
1 TB USB stick
= 40 Mbps

... but 55 hours
Performance = latency?
LATENCY LAGS BANDWIDTH

Recognizing the chronic imbalance between bandwidth and latency, and how to cope with it.
Bandwidth and latency

Transfer time (log scale)

Network round-trip time

Large object

Small object
Bandwidth and latency

Transfer time (log scale)

Network round-trip time

10 GBps, 1 millisecond
- 1 B?
- 100 KB?
- 10 GB?

10 GBps, 100 millisecond
- 1 B?
- 100 KB?
- 10 GB?
Bandwidth and latency

With TCP?

100 KB ≈ 1500 B * (0 + 2 + 4 + 8 + 16 + 32 + 5)

~7 milliseconds in the best case;
0.0143 GBps

Note: you are expected to know how TCP works, e.g.: what is AIMD? bandwidth-delay product?

10 GBps, 1 millisecond
- 1 B?
- 100 KB?
Why Flow-Completion Time is the Right Metric for Congestion Control

Nandita Dukkipati
Computer Systems Laboratory
Stanford University
Stanford, CA 94305-9025
nanditad@stanford.edu

Nick McKeown
Computer Systems Laboratory
Stanford University
Stanford, CA 94305-9025
nickm@stanford.edu

---

When users download a web page, transfer a file, send/read email, or involve the network in almost any interaction, they want their transaction to complete in the shortest time; and it is the main argument of this paper that this is the right metric for congestion control.

Why Flow-Completion Time is the Right Metric

1. WHY WE SHOULD MAKE FLOWS FAST

Users typically want their flows to complete as quickly as possible. Users care about how long it takes to download data, or to complete a transaction. For example, if you are buying a ticket online, you want the transaction to complete quickly.

2. WHY FCT = TIME FROM WHEN THE FIRST PACKET IS SENT IS THE RIGHT METRIC

By FCT, we mean the time from when the first packet of a flow is sent to the time the last packet of a flow is received. This is a natural metric for evaluating the performance of a network.

---

Why Flow-Completion Time is the Right Metric for Congestion Control

1. WHY WE SHOULD MAKE FLOWS FAST

Users typically want their flows to complete as quickly as possible. Users care about how long it takes to download data, or to complete a transaction. For example, if you are buying a ticket online, you want the transaction to complete quickly.

2. WHY FCT = TIME FROM WHEN THE FIRST PACKET IS SENT IS THE RIGHT METRIC

By FCT, we mean the time from when the first packet of a flow is sent to the time the last packet of a flow is received. This is a natural metric for evaluating the performance of a network.

---

Figure 1: Improvement in flow-completion time as link bandwidth increases.
Beyond flow completion time?

- How long does https://www.google.ch/?q=cool+stuff take?
- What’s the best video quality I can watch, without the “buffering”?
- How long does my Hadoop job take?
- ...

Also, we want consistent, predictable performance!
But what about fairness?! 

Suppose a network is flow fair. How useful is that?

"Both the thing being allocated (rate) and what it is allocated among (flows) are completely daft—both unrealistic and impractical."
Food for thought …

How to …

• … translate micro-benchmarks to app-level metrics?
• … make service providers accountable?
• … improve the performance of systems?
• … approach fairness?
Reliability
Three particular considerations with reliability ...

- The end-to-end argument
- The fate-sharing principle
- Packet vs. circuit switching
Three particular considerations with reliability:

- The end-to-end argument
- The fate-sharing principle
- Packet vs. circuit switching
TCP/IP vs. Everything/IP

Initially, TCP/IP was a monolith. Why did we split?
What if no reliable transport is provided?

Every application that needs reliability has to engineer it from scratch

- Programmer burden
- Much higher likelihood of bugs
- Wasteful effort
What if the network layer tried to provide reliable delivery?

Reliable (or unreliable) transport
...built on...

Best-effort global packet delivery
Reliable

#1: My voice call wants speedy delivery, even if it’s lossy …
What if the network layer tried to provide reliable delivery?

Reliable (or unreliable) transport
...built on...

Best-effort global packet delivery
Reliable

#2: Can the network even achieve this at all?
Example: reliably transfer file from host A to B

**Solution 1:**
Check reliability at every step (involving network layer)

**Solution 2:**
Allow unreliable steps (network layer is best-effort)
B checks and tells A to retry on failure
Example: reliably transfer file from host A to B

Solution 1:
Check reliability at every step (involving network layer)

Problem:
Bugs, failures are a truth of life

It’s not **reliable transfer**, if A-B depend on network that can fail
Example: reliably transfer file from host A to B

Solution 2:

Allow unreliable steps (network layer is best-effort).
B checks correctness. On failure, B tells A to retry.
Can still fail, but only if A / B themselves fail.

Solution 2 depends only on what end-points themselves control
Example: reliably transfer file from host A to B

“the end-to-end check of the file transfer application must still be implemented no matter how reliable the communication system becomes”
Question: should we ever implement reliability in the network?
Question: should we ever implement reliability in the network?

Yes, some, to reduce the number of end-end retries needed!

\[ P \text{(retry)} = 1 - 0.90^{10} = 0.65 \]

\[ P \text{(retry)} = 1 - 0.99^{10} = 0.10 \]
Implementing reliability in the network …

… does not reduce end-host complexity

… does increase network complexity

… often imposes overhead for apps that don’t need it

… but can enhance performance in some cases
End-end argument interpretations

1. **Only if sufficient**
   Don’t implement a function at a lower layer unless this is complete

2. **Only if necessary**
   Don’t implement a function at a lower layer unless hosts cannot

3. **Only if useful**
   If hosts can, implement in-network only as an optimization
   BUT only if not burdensome for apps that don’t need it
Beyond a host-centric view ...

Hosts are not the only stakeholders!

Networks want protection from malicious hosts

Things like firewalls are easier to implement in-network
Food for thought ...

Where should we implement ...

• Congestion control?
• Routing?
• Firewalls?
• ...

33
Food for thought ...

• When do we break layering?
• How do we define “end”?
  • TCP vs. application?
  • iPhone vs. iWatch?
  • Distributed applications?
• When do we break the end-to-end argument?
Three particular considerations with reliability ...

- The end-to-end argument
- The fate-sharing principle
- Packet vs. circuit switching
A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

— Leslie Lamport, Microsoft Research

How do we prevent this?
Survivability

End-points should be able to continue communicating without resetting conversation, even under failures.
The fate-sharing principle

When storing state in a distributed system, co-locate it with entities that rely on that state.

State is lost only if those entities fail; then it doesn’t matter.

Example: network connection state at end hosts.
Three particular considerations with reliability ...

- The end-to-end argument
- The fate-sharing principle
- Packet vs. circuit switching
Circuit switching relies on the Resource Reservation Protocol.
One problem with circuit switching is that it doesn’t route around trouble

A is forced to signal a new circuit to restore communication
Pros and cons of circuit switching

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>predictable performance</td>
<td>inefficient if traffic is bursty or short</td>
</tr>
<tr>
<td>simple &amp; fast switching</td>
<td>complex circuit setup/teardown</td>
</tr>
<tr>
<td>once circuit established</td>
<td>which adds delays to transfer</td>
</tr>
<tr>
<td></td>
<td>requires new circuit upon failure</td>
</tr>
</tbody>
</table>
Packet switching routes around trouble

route recomputed on the fly by s2
### Pros and cons of packet switching

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficient use of resources</td>
<td>Unpredictable performance</td>
</tr>
<tr>
<td>Simpler to implement than circuit switching</td>
<td>Requires buffer management and congestion control</td>
</tr>
<tr>
<td>Route around trouble</td>
<td></td>
</tr>
</tbody>
</table>
Packet switching beats Circuit switching with respect to resilience and efficiency.
Conventional wisdom on packets vs. circuits

Reservation makes sense when Peak / Average rate is small
voice traffic has a ratio of 3 or so

Reservation wastes capacity when Peak / Average rate is big
data applications are bursty, ratios >100 are common

Is this true for Internet video?
Perhaps it’s time to rethink reservations vs. on-demand?

Although, today’s streaming implementations are still bursty — fetch a *chunk*, silent, repeat
Weekly reading guide
More on the end-end argument

This “rethink” was in 2001

What has changed?
Why the Internet only just works

M Handley

1. Introduction

The Internet only just works. I want to make it clear though, right from the start, that this is not a forecast of imminent doom and disaster. My reasons for making this assertion are twofold. Firstly, I believe that this has historically been the natural state of the Internet and it is likely to remain so in future. Unless this is understood, then it is hard to understand which problems are really cause for concern, and which we can safely ignore or put off solving till some later date. Secondly, I want to discuss some problems that should be cause for concern, at least in the medium term.

2. The natural state of affairs

If we look back at the history of the Internet, the story is one of constant change. Indeed the phrase ‘Internet time’ is often used to express just how fast things change. But if we look at the core protocols that comprise the Internet at the lower levels of the stack, change has been comparatively slow and carefully considered.

2.1 1970-1993 — a history of change

The first large-scale packet switching network was the ARPAnet, which was used to come to grips with the main architectural issues that would go on to be the basis of the Internet. The basic protocol that underlay the ARPAnet was NCP [1], which combined addressing and transport into a single protocol. Many of the higher-level protocols that would go on to become common on the Internet were first deployed on the ARPAnet. The most obvious are remote log-in, e-mail, and file transfer, but there were also ARPAnet experiments with packet voice, which predate common usage of voice-over-IP by over twenty years.

The ARPAnet was very successful, but it was also clear that flexibility should be of prime importance in the design of a general-purpose successor [2], and as a result reliability was separated from addressing and packet transfer in the design of the Internet protocol suite, with IP being separated from TCP. The switchover to TCP/IP culminated in a flag-day on 1 January 1983 when all remaining ARPAnet nodes switched. There were approximately four hundred nodes; this was probably the last time such a flag-day was possible, and every change since then has needed to be incrementally deployable.

Changing a large network is very difficult. It is much easier to deploy a novel new protocol that fills a void than it is to replace an existing protocol that more or less works. Change is, however, possible when the motivation is sufficient. In 1982 the domain name system (DNS) was deployed, replacing the original hosts.txt file [3] for naming Internet systems. This was a clear response to a scaling problem, but the necessity for change was obvious, and the DNS not only solved the basic issue of distributing files of host names, but also allowed the change to decentralised administration of the namespace. Decentralised administration is one of the basic enablers of the rapid growth of the Internet.

Read the other papers too, if you can!