

Conductor: Adapting with Agents

Mark Yarvis

Dr. Gerald Popok

Dr. Peter Reiher

Adaptation Goals

The Problem:

A wide variety of connectivity options are available, particularly for mobile users.

- Each has different levels of service
 - bandwidth, latency
 - security, real-time guarantees
 - reliability, cost
- Mobile users may experience frequent service changes

However, network applications typically expect a certain level of service and do not degrade gracefully!!!

Need to bridge gap between

- Application requirements and network capabilities
- Transmission cost and user benefit

Common choices:

- QoS: Application reserves required resources prior to communication
- Adaptation: Either reduce application requirements or improve network capabilities



Adaptation is preferred when

- The network is severely deficient
- The user can accept degraded service
- Application transparency is desired
- Network capabilities could be bolstered via software

Approach:

Application transparency

- Don't require application code to be modified
- Use protocol level adaptation

Graceful degradation of applications

- Adapt protocols in a manner that allows a degraded level of application service

Support wide network heterogeneity

- Expect varying characteristics from end to end
- Framework approach allows multi-point deployment

Support for unique user requirements

- Different user requirements lead to different forms of adaptation
- Use of agents allows custom adaptor design

Dynamic deployment

- Don't require prior coordination or registration

Examples:

There are five basic types of adaptation:

- data distillation: e.g., image color or resolution reduction
- pre-fetching: e.g., web page fetch prediction
- caching: e.g., dynamic file mirroring
- prioritization: e.g., ordered reconciliation in file replication
- protocol conversion: allows instant protocol upgrade

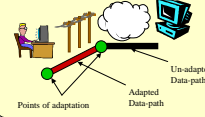
Other examples:

- Server-specified web customization/distillation
- Content-specific data compression
- Real-time user customized e-mail filtering
- Text access to a video conference
- Electronic whiteboards with reduced consistency
- Bulletin board style (zero coordination) file transfer

Deploying Adaptation

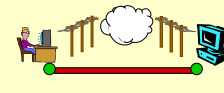
Typical use of adaptation: last mile

- Adaptation occurs around a single "bad" link
- Proxy deployed at entry to "good" network
- Transparency requires two points of adaptation, but one can be on client host



Single proxy is sufficient (for this architecture, however ...)

More complex architectures require more flexibility:



- Consider a connection which crosses two low bandwidth links.
- We could deploy compression across each link, but prefer end-to-end adaptation.

- In this case, end-to-end adaptation leads to
 - Greater efficiency
 - Better results (consider lossy compression)

- However, end-to-end adaptation fails the following considerations
 - Security constraints
 - Responsiveness
 - Load balancing



- Consider a connection with conflicting attributes:
 - Low bandwidth (modem): Use compression or filter
 - High latency (satellite): Use pre-fetching

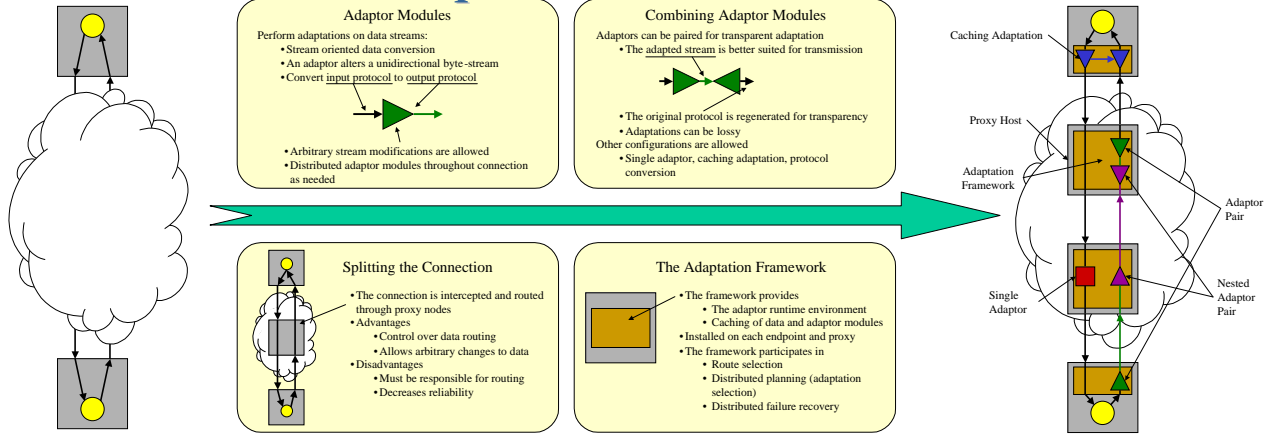
These adaptations cannot be combined end-to-end.

Requires at least three points of adaptation.

For even more complex architectures, many more points of adaptation may be required:



Adaptation Architecture



Adaptation and Reliability

Why does adaptation make reliability difficult?

Additional points of failure:

- Proxy failure leads to connection failure
- Reliable links only provide failure detection

Adaptation breaks usual reliability assumption:

- Data transmitted ≠ data received!!!
- Retransmission required adaptor state

Adaptors are combined in an inter-dependent manner

Reliability requirements:

- Must be independent of data attributes (e.g., length)
 - segmentation
- Can't require proxy health
 - end-to-end segment recovery
- Must preserve adaptor pairing and consistent data typing
 - adaptation recovery algorithm

Segmentation

Consider an adaptor that abbreviates words in a sentence.

- The output stream is shorter than the input stream (perhaps at least entry points)
- If a failure occurs, what data must be retransmitted?

We must segment data to preserve "equivalence" between input and output streams:

- Preserves the notion of what data has been delivered

Segment Recovery

Segment Creation

- Initially, the stream contains one-byte segments
- Adaptors combine segments whenever cross-segment modifications are made
- Segments grow, but do not shrink

Caching

- All segments must be cached at the source
- Additional caches may be used along the stream (perhaps at least entry points)
- Acknowledgements are provided

Recovery

- Each framework keeps track of segments that have been seen, but not acknowledged.
- When a failure occurs, frameworks on either side of the failure compare lists, and request retransmission of outstanding segments
- Retransmission occurs from the closest cache
- Retransmitted segments can be freely re-adapted

Adaptation Recovery

Adaptors are interdependent

- Pairs of adaptors must fail in pairs
- Failure of an adaptor pair will change the input type to sub-adaptors

Recovery modes

- When a single adaptor fails, consistency can be maintained by
 - removing dependent adaptors, or
 - replacing the failed adaptor
- Since adaptors have state, it is easier to destroy than create

Recovery algorithm

- Need recovery without global knowledge
- At each point in the stream, keep track of the "hierarchy" of adaptations present.
- When failure occurs, adjacent frameworks compare "hierarchies"
- The "test common adaptations" between hierarchies remains, all others are removed.

Distributed Planning

Issues:

Local planning cannot always be correct

- These costs reduce potential gain

How can this node know to prefer end-to-end adaptation?

Changing adaptations is expensive

- Use global planning

But, partitions are likely

- Use partition-wide planning

Planning Algorithm:

1. A node identifies a significant change in local state
 - E.g. link capacity, local load
2. A request for planning is forwarded toward one endpoint
 - Identifies partitions
 - Coordinates multiple requests
3. Each node sends local status info toward other endpoint
 - Neighbor-to-neighbor transmission identifies partitions
 - Algorithm begins here for new connections
4. Information from all nodes is used to generate a new plan
5. Plan is distributed to all nodes
 - Adaptation changes can be treated as failures

Information Gathering Algorithm:

Each node has

- An ID
- Local status
- A local status version number
- A status cache

A node caches

- Status of other nodes along each connection
- The list of nodes to which each status has already been forwarded

During information gathering

- Only forward status of a node if
 - Its status has not previously been forwarded.
 - Its status version has changed

To form new connection A-C-D-E, D must transmit A's status to E

Information gathering is costly

- Particularly when bandwidth and latency are already an issue
- These costs reduce potential gain

Exploit properties of node status:

- Changes infrequently
- Independent of individual connections

Exploit properties of connections:

- Routing paths frequently overlap
- The same proxy nodes are frequently adjacent

Amortize cost of information gathering across many connections

Two connections: A-C-D-F and B-C-D-E

