Push it - update 2: a P2P protocol for Append-Only Push (AOP)

Christian Tschudin, U of Basel, Switzerland

ICNRG interim meeting in Macao, China
September 27, 2019
Accumulative information, items typically named by some hash

Global broadcast-only semantics: novelty is replicated everywhere, eventually

History:
- Sep 2018 / panel at ICN18
- Mar 2019 / ICNRG Prague: broadcast-only
- Jul 2019 / ICNRG Montreal, update 1: problems of pull (e.g., “recursion corridor”)

Today’s update 2: zoom-in to the protocol level
Overview

1. Recap: Secure Scuttlebutt’s append-only logs
2. Logical design of a replication protocol
3. Two implementation styles: pullified vs pushified
4. AOP - a pushified replication protocol
5. A surprise guest
6. Status and Conclusions
1) Append-only logs (SSB fame)

- Producer ID
  = public key of a key pair

- Append-only log
  = hash chain of signed events

- Task of the replication layer:
  - propagate novelty unconditionally
  - often called “push”
Given: Two nodes N1 and N2 with their sets of logs.

Replication task when N1 and N2 peer:
1’) Append-Only logs

Given: Two nodes N1 and N2 with their sets of logs

Replication task when N1 and N2 peer:
1’) Append-Only logs

Given: Two nodes N1 and N2 with their sets of logs

Replication task when N1 and N2 peer:

- To “level out” novelty
  - any log extensions that N1 has but N2 is lacking, must be copied to N2
  - and vice versa
1’) Append-Only logs

Given: Two nodes N1 and N2 with their sets of logs

Replication task when N1 and N2 peer:

- To “level out” **novelty**
  - any log extensions that N1 has but N2 is lacking, must be copied to N2
  - and vice versa
1’) Append-Only logs

Given: Two nodes N1 and N2 with their sets of logs

Replication task when N1 and N2 peer:

- To “level out” novelty
  - any log extensions that N1 has but N2 is lacking, must be copied to N2
  - and vice versa

- Applies to the intersection of the log sets
2) AOP - logical design

à la FTP (a replication protocol):
separate control and data channels:

① **FTP Client** opens command channel to **FTP Server** and requests “passive” mode

② **FTP Server** allocates port for the data channel and transmits the port number to use for data transmission

③ **FTP Client** opens the data channel on the specified port
2) AOP - logical design

à la FTP (a replication protocol): separate control and data channels:

- **Control dialogue**
  - configuration
  - commands
  - status

1. FTP Client opens command channel to FTP Server and requests "passive" mode
2. FTP Server allocates port for the data channel and transmits the port number to use for data transmission
3. FTP Client opens the data channel on the specified port
2) AOP - logical design

à la FTP (a replication protocol):
separate control and data channels:

- **Control dialogue**
  - configuration
  - commands
  - status

- **Data**
  actual transfer of information
2’) AOP - logical design

AOP = Append-only Push  // or: “Append-only (replication) Protocol”, or …

Control verbs:
2’) AOP - logical design

AOP = Append-only Push  // or: “Append-only (replication) Protocol”, or …

Control verbs:

HELLO my_id=N1 dh=%#$  # handshake msgs, incl DH negotiation
2’) AOP - logical design

AOP = Append-only Push     // or: “Append-only (replication) Protocol”, or …

Control verbs:

HELLO my_id=N1 dh=%#$     # handshake msgs, incl DH negotiation...
PORT udp=1.2.3.4/567      # configuration details
2’) AOP - logical design

AOP = Append-only Push     // or: “Append-only (replication) Protocol”, or …

Control verbs:

HELLO my_id=N1 dh=%#$       # handshake msgs, incl DH negotiation
PORT udp=1.2.3.4/567        # configuration details
CREDIT 4                    # flow control (back pressure)
AOP = Append-only Push     // or: “Append-only (replication) Protocol”, or …

Control verbs:

HELLO my_id=N1 dh=%#$     # handshake msgs, incl DH negotiation
PORT udp=1.2.3.4/567      # configuration details
CREDIT 4                  # flow control (back pressure)
WANT B:5 credit=2         # I have B:4, send anything newer
2’) AOP - logical design

AOP = Append-only Push       // or: “Append-only (replication) Protocol”, or …

Control verbs:

HELLO my_id=N1 dh=%#$          # handshake msgs, incl DH negotiation
PORT udp=1.2.3.4/567           # configuration details
CREDIT 4                        # flow control (back pressure)
WANT B:5 credit=2              # I have B:4, send anything newer
WANT C:7                       # I have C:6, send anything newer
WANT ...                       # many more WANTs declarations
2’) AOP - logical design

AOP = Append-only Push     // or: “Append-only (replication) Protocol”, or …

Control verbs:

HELLO my_id=N1 dh=%#$     # handshake msgs, incl DH negotiation…
PORT udp=1.2.3.4/567       # configuration details
CREDIT 4                    # flow control (back pressure)
WANT B:5 credit=2          # I have B:4, send anything newer
WANT C:7                   # I have C:6, send anything newer
WANT ...                   # many more WANTs declarations
HAVE A:3                   # optional: announce log set
2’) AOP - logical design

AOP = Append-only Push // or: “Append-only (replication) Protocol”, or …

Control verbs:

HELLO my_id=N1 dh=%#$ # handshake msgs, incl DH negotiation…
PORT udp=1.2.3.4/567 # configuration details
CREDIT 4 # flow control (back pressure)
WANT B:5 credit=2 # I have B:4, send anything newer
WANT C:7 # I have C:6, send anything newer
WANT ... # many more WANTs declarations
HAVE A:3 # optional: announce log set
2’) AOP - logical design

AOP = Append-only Push  // or: “Append-only (replication) Protocol”, or …

Control verbs:

HELLO my_id=N1 dh=%#$  # handshake msgs, incl DH negotiation…
PORT udp=1.2.3.4/567    # configuration details
CREDIT 4                 # flow control (back pressure)
WANT B:5 credit=2        # I have B:4, send anything newer
WANT C:7                 # I have C:6, send anything newer
WANT ...                 # many more WANTs declarations
HAVE A:3                 # optional: announce log set
2”) AOP - logical design

Show time-sequence diagram here, and ports ...

1. FTP Client opens command channel to FTP Server and requests "passive" mode.
2. FTP Server allocates port for the data channel and transmits the port number to use for data transmission.
3. FTP Client opens the data channel on the specified port.
3) **Pullified vs Pushified replication**

Pullified implementation style:

- “mainstream”, client/server mindset, RPC
- chosen by NDN, SSB (!)

Pushified style:

- See *later* in this slide set.
- Note: AOP is **not** SSB (yet)
3) Pullified vs Pushified replication

Pullified implementation style:

- “mainstream”, client/server mindset, RPC
- chosen by NDN, SSB (!)

Pushified style:

- See later in this slide set.
- Note: AOP is not SSB (yet)
3’) Pullified vs Pushified replication

Pullified implementation style:

- “mainstream”, client/server mindset, RPC
- chosen by NDN, SSB (!)

SSB:

```
WANT C:5, credit=2 ->
RPC createStream(id=C,seq=5,max=2)
```

Overall backpressure (the CREDIT verb): via underlying TCP stream
3”) One Problem of Pullification

In SSB:

SSB:

WANT \textit{C:5, credit=2} →
RPC \texttt{createStream(id=C,seq=5,max=2)}

“secure hand-shake” protocol
3”\(^{\text{3}}\) One Problem of Pullification

In SSB:

- At peering time, potentially (and in practice) \textit{thousands} of RPC requests
- A nuisance for user end nodes that often have only one log with novelty

SSB:

\texttt{WANT C:5, credit=2 ->}
\texttt{RPC createStream(id=C,seq=5,max=2)}
3”) One Problem of Pullification

In SSB:

• At peering time, potentially (and in practice) **thousands** of RPC requests

• A nuisance for user end nodes that often have only one log with novelty

In NDN:

• Must **repeatedly re-issue** the WANT LLI (long-lived interest) because peer could have crashed. This will also be hundreds or thousands LLIs, in the future
4) *Pushified* replication in AOP

Main idea:
- nodes append their WANT items to separate logs (W1, W2)
- these “WANT logs” *being replicated* like all others logs = “caching”
- but not replicated beyond the peer

before crash

after crash
4) **Pushified** replication in AOP

Main idea:
- nodes append their WANT items to separate logs (W1, W2)
- these “WANT logs” being replicated like all others logs = “caching”
- but not replicated beyond the peer

Advantage: “free” recovery after crash, or at a future peering time

Reminder: log W2 contains all log IDs that node N2 wants
4) **Pushified replication in AOP**

Main idea:
- nodes append their WANT items to separate logs (W1, W2)
- these “WANT logs” being replicated like all others logs = “caching”
- but not replicated beyond the peer

Advantage: “free” recovery after crash, or at a future peering time

**Reminder:** log W2 contains all log IDs that node N2 wants

before crash:

```
-> HELLO id=N1, want_id=W1
<- HELLO id=N2, want_id=W2

-> WANT W2:1

<- W2:1 (~ WANT B:5)
<- W2:2 (~ WANT C:7)
...```

after crash
4) **Pushified replication in AOP**

**Main idea:**
- nodes append their WANT items to separate logs (W1, W2)
- these “WANT logs” being replicated like all others logs = “caching”
- but not replicated beyond the peer

**Advantage:** “free” recovery after crash, or at a future peering time

**Reminder:** log W2 contains all log IDs that node N2 wants
5) A surprise guest: TCP

We do some relabelling:
5) A surprise guest: TCP

We do some relabelling:

• HELLO  
becomes TCP’s 3-way handshake

HELLO my_id=N1, want_id=C:25, have_id=D:78
5) A surprise guest: TCP

We do some relabelling:

- **HELLO**  
  becomes TCP’s 3-way handshake

  HELLO my_id=N1, want_id=C:25, have_id=D:78

- **tcp_ack=34**  
  → WANT C:35
5) A surprise guest: TCP

We do some relabelling:

• **HELLO**
  becomes TCP’s 3-way handshake

  HELLO my_id=N1, want_id=C:25, have_id=D:78

• **tcp_ack=34**  →  WANT C:35  →

• **tcp_seq=44**  →  HAVE D:44
5) A surprise guest: TCP

We do some relabelling:

• **HELLO** becomes TCP’s 3-way handshake

  \[\text{HELLO my_id=N1, want_id=C:25, have_id=D:78}\]

• **tcp_ack=34** -> **WANT C:35** ->

• **tcp_seq=44** -> **HAVE D:44**
5) A surprise guest: TCP

We do some relabelling:

• **HELLO**  
  becomes TCP’s 3-way handshake

  HELLO  my_id=N1, want_id=C:25, have_id=D:78

• **tcp_ack=34**  
  -> WANT  C:35  ->

• **tcp_seq=44**  
  -> HAVE  D:44

• Cumulative ACK: in TCP and AOP

TCP:

ID=<src,dst>, seq=X, ack=Y,  
[optional data byte (events)]
5) A surprise guest: TCP

We do some relabelling:

- **HELLO** becomes TCP’s 3-way handshake
  
  HELLO my_id=N1, want_id=C:25, have_id=D:78

- **tcp_ack=34** -> WANT C:35 ->

- **tcp_seq=44** -> HAVE D:44

- Cumulative ACK: in TCP and AOP

Not a suprise, really: TCP is a “replication protocol”, can also be called a “controlled push” (=sender driven, flow-controlled)
5’) A surprise guest: TCP

TCP … in comparison to NDN and AOP
5’) A surprise guest: TCP

TCP … in comparison to NDN and AOP

• NDN “pulls content via (TCP’s) ACK”
  - has credit=1
  - lacks cumulative ACK
    (together this feature is called “flow balance”)
  - have to use parallel Interests to fill the pipeline
5’) A surprise guest: TCP

TCP … in comparison to NDN and AOP

• NDN “pulls content via (TCP’s) ACK”
  - has credit=1
  - lacks cumulative ACK
    (together this feature is called “flow balance”)
  - have to use parallel Interests to fill the pipeline

• AOP more like TCP
  - “stream” thinking, cumulative ack
  - both remember information frontier (packet loss)
  - difference to TCP: AOP supports *multiple* streams,
    AOP can *resume* its streaming after a node crash, hides “Internet weather”
6) Status and Conclusions

AOP is a pushified version of a replication protocol for event streams

- AOP is not SSB: perhaps SSB will adopt it?
- AOP is not a general pub/sub:
  - strict (crypto-enforced) log discipline
  - reliable
  - producer-centric (e.g., no N:1 sending to a “topic channel”)
- AOP is not TCP, but includes similar mindset
AOP is a pushified version of a replication protocol for event streams

- AOP is *not* SSB: perhaps SSB will adopt it?

- AOP is *not* a general pub/sub:
  - strict (crypto-enforced) log discipline
  - reliable
  - producer-centric (e.g., no N:1 sending to a “topic channel”)

- AOP is *not* TCP, but includes similar mindset

AOP: running Python Proof-of-Concept for connection-less settings (UDP, ethernet)